

# **Le Manuel de Référence Canonique de Csound**

**Version 5.12**

**Barry Vercoe, MIT Media Lab  
et. al.**

---

# **Le Manuel de Référence Canonique de Csound: Version 5.12**

par Barry Vercoe et et. al.

---

---

---

# Table des matières

Préface .....	xxix
Préface du Manuel de Csound .....	xxix
Histoire du Manuel de Référence Canonique de Csound .....	xxx
Mentions de copyright .....	xxxi
Débuter avec Csound .....	xxxiii
Les nouveautés de Csound 5.12 .....	xxxv
I. Vue d'Ensemble .....	1
Introduction .....	4
Développements Récents .....	5
Caractéristiques de Csound 5 .....	5
Caractéristiques de CsoundAC .....	6
La commande Csound .....	9
Ordre de priorité .....	9
Description de la syntaxe de la commande .....	9
Ligne de Commande de Csound .....	11
Options de Ligne de Commande (par Catégorie) .....	21
Variables d'Environnement de Csound .....	31
Format de Fichier Unifié pour les Orchestres et les Partitions .....	34
Description .....	34
Exemple .....	36
Fichier de Paramètres de Ligne de Commande (.csoundrc) .....	37
Prétraitement du Fichier Partition .....	37
La Fonction Extract .....	37
Prétraitement Indépendant avec Scsort .....	38
Utiliser Csound .....	39
Sortie Console de Csound .....	39
Comment Csound5 fonctionne .....	40
Valeurs d'amplitude dans Csound .....	41
Audio en temps-réel .....	43
Entrées/Sorties en temps-réel sur Linux .....	44
Windows .....	49
Mac .....	50
Optimisation de la Latence Audio en E/S .....	50
Configuration .....	52
Syntaxe de l'Orchestre .....	53
Instructions de l'En-tête de l'Orchestre .....	54
Instructions de Bloc d'Instrument et d'Opcodes .....	54
Instructions Ordinaires .....	55
Types, Constantes et Variables .....	55
Initialisation de Variable .....	57
Expressions .....	57
Répertoires et Fichiers .....	57
Nomenclature .....	58
Macros .....	58
Instruments Nommés .....	59
Opcodes Définis par l'Utilisateur (UDO) .....	61
La Partition Numérique Standard .....	63
Prétraitement des Partitions Standard .....	63
Carry .....	63
Tempo .....	64
Sort .....	64
Instructions de Partition .....	65
Symboles Next-P et Previous-P .....	65



Ramping .....	66
Macros de Partition .....	67
Partition dans Plusieurs Fichiers .....	69
Evaluation des Expressions .....	70
Chaînes de caractères dans les p-champs .....	71
Frontaux .....	73
CsoundAC .....	74
CsoundVST .....	75
TclCsound .....	78
L'interpréteur Tcl : cstclsh .....	78
Cswish: le shell de fenêtrage .....	78
Un serveur Csound .....	79
Un Environnement de Scripting .....	80
TclCsound comme encapsuleur de langage .....	81
Référence des Commandes de TclCsound .....	81
Construire Csound .....	84
Liens Csound .....	90
II. Vue d'Ensemble des Opcodes .....	91
Générateurs de Signal .....	95
Synthèse/Resynthèse Additive .....	95
Oscillateurs Elémentaires .....	95
Oscillateurs à Spectre Dynamique .....	95
Synthèse FM .....	96
Synthèse Granulaire .....	96
Synthèse Hyper Vectorielle .....	97
Générateurs Linéaires et Exponentiels .....	97
Générateurs d'Enveloppe .....	98
Modèles et Emulations .....	98
Phaseurs .....	99
Générateurs de Nombres Aléatoires (de Bruit) .....	100
Reproduction de Sons Echantillonnés .....	101
Soundfonts .....	101
Synthèse par Balayage .....	103
Accès aux Tables .....	104
Synthèse par Terrain d'Ondes .....	105
Modèles Physiques par Guide d'Onde .....	105
Entrée et Sortie de Signal .....	106
Entrées et Sorties Fichier .....	106
Entrée de Signal .....	106
Sortie de Signal .....	106
Bus Logiciel .....	107
Impression et Affichage .....	107
Requêtes sur les Fichiers Sons .....	107
Modificateurs de Signal .....	109
Modificateurs d'Amplitude et Traitement des Dynamiques .....	109
Convolution et Morphing .....	109
Retard .....	109
Panning et Spatialisation .....	110
Réverbération .....	112
Opérateurs du Niveau Echantillon .....	112
Limiteurs de Signal .....	113
Effets Spéciaux .....	113
Filtres Standard .....	113
Filtres Spécialisés .....	115
Guides d'Onde .....	115
Distorsion Non-Linéaire et Distorsion de Phase .....	115
Contrôle d'Instrument .....	117
Contrôle d'Horloge .....	117

Valeurs Conditionnelles .....	117
Instructions de Contrôle de Durée .....	117
Widgets FLTK et contrôleurs GUI .....	117
Conteneurs FLTK .....	120
Valuateurs FLTK .....	120
Autres Widgets FLTK .....	121
Modifier l'Apparence des Widgets FLTK .....	122
Opcodes Généraux relatifs aux Widgets FLTK .....	122
Appel d'Instrument .....	123
Contrôle Séquentiel d'un Programme .....	123
Contrôle de l'Exécution en Temps Réel .....	124
Initialisation et Réinitialisation .....	124
Détection et Contrôle .....	125
Piles .....	126
Contrôle de sous-instrument .....	126
Lecture du Temps .....	127
Contrôle des Tables de Fonction .....	128
Requêtes sur une Table .....	128
Opérations de Lecture/Ecriture de Table .....	128
Lecture de Table avec Sélection Dynamique .....	129
Opérations Mathématiques .....	130
Conversion d'Amplitude .....	130
Opérations Arithmétiques et Logiques .....	130
Comparateurs et Accumulateurs .....	130
Fonctions Mathématiques .....	131
Opcodes Equivalents à des Fonctions .....	131
Fonctions aléatoires .....	132
Fonctions Trigonométriques .....	132
Opcodes d'Algèbre Linéaire .....	133
Conversion des Hauteurs .....	143
Fonctions .....	143
Opcodes de Hauteurs .....	143
Support MIDI en Temps-Réel .....	144
Clavier Virtuel MIDI .....	145
Entrée MIDI .....	148
Sortie de Message MIDI .....	148
Entrée et Sortie Génériques .....	149
Convertisseurs .....	149
Extension d'Evènements .....	149
Sortie de Note-on/Note-off .....	149
Opcodes pour l'Interopérabilité MIDI/Partition .....	150
Messages System Realtime .....	151
Banques de Réglettes .....	151
Traitement Spectral .....	153
Resynthèse par Transformée de Fourier à Court-Terme (STFT) .....	153
Resynthèse par Codage Prédicatif Linéaire (LPC) .....	154
Traitement Spectral Non-standard .....	154
Outils pour le Traitement Spectral en Temps Réel (opcodes pvs) .....	154
Traitement Spectral avec ATS .....	156
Opcodes Loris .....	156
Chaînes de Caractères .....	161
Opcodes de Manipulation de Chaîne .....	162
Opcodes de Conversion de Chaîne .....	162
Opcodes Vectoriels .....	164
Opérateurs de Tableaux de Vecteurs .....	164
Opérations Entre un Signal Vectoriel et un Signal Scalaire .....	164
Opérations Entre deux Signaux Vectoriels .....	165
Générateurs Vectoriels d'Enveloppe .....	165

Limitation et Enroulement des Signaux Vectoriels de Contrôle .....	166
Chemins de Retard Vectoriel au Taux de Contrôle .....	166
Générateurs de Signal Aléatoire Vectoriel .....	166
Système de Patch Zak .....	168
Accueil de Plugin .....	169
DSSI et LADSPA pour Csound .....	169
VST pour Csound .....	169
OSC et Réseau .....	171
OSC .....	171
Réseau .....	171
Opcodes pour le Traitement à Distance .....	171
Opcodes Mixer .....	172
Opcodes de Graphe de Fluence .....	173
Opcodes Python .....	176
Introduction .....	176
Syntaxe de l'Orchestre .....	176
Opcodes pour le traitement d'image .....	178
Opcodes divers .....	179
III. Référence .....	180
Opcodes et Opérateurs de l'Orchestre .....	203
!= .....	204
#define .....	206
#include .....	210
#undef .....	212
#ifdef .....	213
#ifndef .....	215
\$NOM .....	216
% .....	219
&& .....	221
> .....	222
>= .....	224
< .....	226
<= .....	228
* .....	230
+ .....	232
- .....	234
/ .....	236
= .....	238
== .....	240
^ .....	242
.....	244
0dbfs .....	245
<< .....	248
>> .....	250
& .....	251
.....	253
¬ .....	254
# .....	255
a .....	256
abetarand .....	258
abexprnd .....	259
abs .....	260
acauchy .....	262
active .....	263
adsr .....	267
adsyn .....	270
adsynt .....	272
adsynt2 .....	275

aexprand .....	277
aftouch .....	278
agauss .....	280
agogobel .....	281
alinrand .....	282
alpass .....	283
alwayson .....	286
ampdb .....	287
ampdbfs .....	289
ampmidi .....	291
apcauchy .....	293
apoisson .....	294
apow .....	295
areson .....	296
aresonk .....	298
atone .....	299
atonek .....	301
atonex .....	302
atrirand .....	303
ATSadd .....	304
ATSaddnz .....	306
ATSbufread .....	308
ATScross .....	310
ATSinfo .....	312
ATSinterpread .....	314
ATSread .....	315
ATSreadnz .....	317
ATSpartialtap .....	319
ATSSinnoi .....	320
aunirand .....	322
aweibull .....	323
babo .....	324
balance .....	328
bamboo .....	330
barmodel .....	332
bbcutm .....	334
bbcuts .....	339
betarand .....	341
bexprnd .....	343
bformenc .....	345
bformenc1 .....	347
bformdec .....	349
bformdec1 .....	351
binit .....	353
biquad .....	354
biquada .....	358
birnd .....	359
bqrez .....	361
butbp .....	363
butbr .....	364
buthp .....	365
butlp .....	366
butterbp .....	367
butterbr .....	369
butterhp .....	371
butterlp .....	373
button .....	375
buzz .....	376

cabasa .....	378
cauchy .....	380
ceil .....	382
cent .....	383
cggoto .....	385
chanctrl .....	387
changed .....	388
chani .....	390
chano .....	391
chebyshevpoly .....	392
checkbox .....	395
chn .....	397
chnclear .....	399
chnexport .....	400
chnget .....	402
chnmix .....	404
chnparams .....	405
chnrecv .....	406
chnsend .....	408
chnset .....	410
chuap .....	412
cigoto .....	416
ckgoto .....	418
clear .....	420
clfilt .....	421
clip .....	424
clock .....	427
clockoff .....	428
clockon .....	429
cngoto .....	430
comb .....	432
compress .....	435
connect .....	437
control .....	438
convle .....	439
convolve .....	440
cos .....	443
cosh .....	445
cosinv .....	447
cps2pch .....	449
cpsmidi .....	453
cpsmidib .....	455
cpsmidinn .....	457
cpsoct .....	460
cpspch .....	463
cpstmid .....	466
cpstun .....	469
cpstuni .....	472
cpsxpch .....	475
cpuprc .....	479
cross2 .....	481
crossfm .....	483
crunch .....	486
ctrl14 .....	488
ctrl21 .....	490
ctrl7 .....	492
ctrlinit .....	494
cuserrnd .....	495

dam .....	497
date .....	501
dates .....	503
db .....	505
dbamp .....	507
dbfsamp .....	509
dcblock .....	511
dcblock2 .....	513
dconv .....	514
delay .....	516
delay1 .....	518
delayk .....	519
delayr .....	521
delayw .....	522
deltap .....	524
deltap3 .....	526
deltapi .....	528
deltapn .....	530
deltapx .....	532
deltapxw .....	534
denorm .....	536
diff .....	537
diskgrain .....	539
diskin .....	542
diskin2 .....	545
dispfft .....	548
display .....	550
distort .....	552
distort1 .....	554
divz .....	556
doppler .....	558
downsamp .....	560
dripwater .....	562
dssiactivate .....	564
dssiaudio .....	565
dssictls .....	566
dssiinit .....	567
dssilist .....	569
dumpk .....	570
dumpk2 .....	572
dumpk3 .....	574
dumpk4 .....	576
dusernd .....	578
else .....	580
elseif .....	581
endif .....	582
endin .....	583
endop .....	585
envlpx .....	586
envlpxr .....	589
ephasor .....	591
eqfil .....	592
event .....	594
event_i .....	597
exitnow .....	599
exp .....	600
expcurve .....	602
expon .....	604

exprand .....	606
expseg .....	608
expsega .....	610
expsegr .....	612
ficlose .....	615
filebit .....	617
filelen .....	619
filenchnls .....	621
filepeak .....	623
filesr .....	625
filter2 .....	627
fin .....	629
fini .....	631
fink .....	633
fiopen .....	635
flanger .....	637
flashtxt .....	639
FLbox .....	641
FLbutBank .....	646
FLbutton .....	649
FLcloseButton .....	654
FLcolor .....	657
FLcolor2 .....	659
FLcount .....	660
FLexecButton .....	663
FLgetsnap .....	666
FLgroup .....	668
FLgroupEnd .....	670
FLgroupEnd .....	671
FLhide .....	672
FLhvsBox .....	673
FLhvsBoxSetValue .....	674
FLjoy .....	675
FLkeyIn .....	678
FLknob .....	680
FLlabel .....	685
FLloadsnap .....	687
FLmouse .....	688
flooper .....	690
flooper2 .....	692
floor .....	694
FLpack .....	695
FLpackEnd .....	698
FLpack_end .....	699
FLpanel .....	700
FLpanelEnd .....	703
FLpanel_end .....	704
FLprintk .....	705
FLprintk2 .....	706
FLroller .....	707
FLrun .....	710
FLsavesnap .....	711
FLscroll .....	716
FLscrollEnd .....	719
FLscroll_end .....	720
FLsetAlign .....	721
FLsetBox .....	722
FLsetColor .....	724

FLsetColor2 .....	726
FLsetFont .....	727
FLsetPosition .....	729
FLsetSize .....	730
FLsetsnap .....	731
FLsetSnapGroup .....	733
FLsetText .....	734
FLsetTextColor .....	736
FLsetTextSize .....	737
FLsetTextType .....	738
FLsetVal_i .....	741
FLsetVal .....	742
FLshow .....	743
FLslidBnk .....	744
FLslidBnk2 .....	748
FLslidBnkGetHandle .....	751
FLslidBnkSet .....	752
FLslidBnkSetk .....	753
FLslidBnk2Set .....	755
FLslidBnk2Setk .....	756
FLslider .....	759
FLtabs .....	765
FLtabsEnd .....	770
FLtabs_end .....	771
FLtext .....	772
FLupdate .....	775
fluidAllOut .....	776
fluidCCi .....	779
fluidCCk .....	780
fluidControl .....	781
fluidEngine .....	783
fluidLoad .....	787
fluidNote .....	789
fluidOut .....	791
fluidProgramSelect .....	793
fluidSetInterpMethod .....	795
FLvalue .....	796
FLvkeybd .....	798
FLvslidBnk .....	799
FLvslidBnk2 .....	803
FLxyin .....	806
fmb3 .....	809
fmbell .....	812
fmmetal .....	815
fmpercfl .....	818
fmrhode .....	821
fmvoice .....	824
fmwurlie .....	826
fof .....	829
fof2 .....	832
fofilter .....	838
fog .....	840
fold .....	842
follow .....	844
follow2 .....	846
foscil .....	848
foscili .....	850
fout .....	852



fouti .....	857
foutir .....	859
foutk .....	861
fprintks .....	863
fprints .....	869
frac .....	871
freeverb .....	873
ftchnls .....	875
ftconv .....	877
ftfree .....	880
ftgen .....	881
ftgenonce .....	884
ftgentmp .....	886
ftlen .....	888
ftload .....	890
ftloadk .....	891
ftlptim .....	892
ftmorf .....	894
ftsav .....	896
ftsavk .....	898
ftsr .....	899
gain .....	901
gainslider .....	903
gauss .....	905
gbuzz .....	907
getcfcg .....	910
gogobel .....	911
goto .....	913
grain .....	915
grain2 .....	917
grain3 .....	921
granule .....	926
guiro .....	930
harmon .....	932
harmon2 .....	935
hilbert .....	937
hrtfer .....	941
hrtfmove .....	943
hrtfmove2 .....	946
hrtfstat .....	949
hsboscil .....	952
hvs1 .....	955
hvs2 .....	959
hvs3 .....	965
i .....	968
ibetarand .....	969
ibexprnd .....	970
icauchy .....	971
ictrl14 .....	972
ictrl21 .....	973
ictrl7 .....	974
iexprand .....	975
if .....	976
igauss .....	981
igoto .....	982
ihold .....	984
ilinrand .....	986
imagecreate .....	987

imagefree .....	989
imagegetpixel .....	991
imageload .....	993
imagesave .....	995
imagesetpixel .....	997
imagesize .....	999
imidic14 .....	1001
imidic21 .....	1002
imidic7 .....	1003
in .....	1004
in32 .....	1005
inch .....	1006
inh .....	1007
init .....	1008
initc14 .....	1009
initc21 .....	1010
initc7 .....	1011
inleta .....	1012
inletk .....	1013
inletf .....	1014
ino .....	1015
inq .....	1016
inrg .....	1017
ins .....	1018
insremot .....	1019
insglobal .....	1021
instimek .....	1022
instimes .....	1023
instr .....	1024
int .....	1027
integ .....	1029
interp .....	1031
invalue .....	1034
inx .....	1035
inz .....	1036
ioff .....	1037
ion .....	1038
iondur .....	1039
iondur2 .....	1040
ioutat .....	1041
ioutc .....	1042
ioutc14 .....	1043
ioutpat .....	1044
ioutpb .....	1045
ioutpc .....	1046
ipcauchy .....	1047
ipoisson .....	1048
ipow .....	1049
is16b14 .....	1050
is32b14 .....	1051
islider16 .....	1052
islider32 .....	1053
islider64 .....	1054
islider8 .....	1055
itablecopy .....	1056
itablegpw .....	1057
itablemix .....	1058
itablew .....	1059

itrirand .....	1060
iunirand .....	1061
iweibull .....	1062
jacktransport .....	1063
jitter .....	1065
jitter2 .....	1067
jspline .....	1069
k .....	1070
kbetarand .....	1071
kbexprnd .....	1072
kcauchy .....	1073
kdump .....	1074
kdump2 .....	1075
kdump3 .....	1076
kdump4 .....	1077
kexprand .....	1078
kfilter2 .....	1079
kgauss .....	1080
kgoto .....	1081
klinrand .....	1083
kon .....	1084
koutat .....	1085
koutc .....	1086
koutc14 .....	1087
koutpat .....	1088
koutpb .....	1089
koutpc .....	1090
kpcauchy .....	1091
kpoisson .....	1092
kpow .....	1093
kr .....	1094
kread .....	1095
kread2 .....	1096
kread3 .....	1097
kread4 .....	1098
ksmps .....	1099
ktableseg .....	1100
ktirand .....	1101
kunirand .....	1102
kweibull .....	1103
lfo .....	1104
limit .....	1106
line .....	1107
linen .....	1109
linenr .....	1111
lineto .....	1112
linrand .....	1113
linseg .....	1115
linsegr .....	1118
locsend .....	1121
locsig .....	1123
log .....	1126
log10 .....	1128
logbtwo .....	1130
logcurve .....	1132
loop_ge .....	1134
loop_gt .....	1135
loop_le .....	1136

loop_lt .....	1137
loopseg .....	1138
loopsegp .....	1140
looptseg .....	1142
loopxseg .....	1144
lorenz .....	1146
lorisread .....	1149
lorismorph .....	1151
lorisplay .....	1152
loscil .....	1153
loscil3 .....	1156
loscilx .....	1159
lowpass2 .....	1160
lowres .....	1162
lowresx .....	1164
lpf18 .....	1166
lpfreson .....	1168
lphasor .....	1169
lpinterp .....	1171
lposcil .....	1172
lposcil3 .....	1173
lposcila .....	1174
lposcilsa .....	1175
lposcilsa2 .....	1176
lpread .....	1177
lpreson .....	1179
lpshold .....	1180
lpsholdp .....	1182
lpslot .....	1183
mac .....	1185
maca .....	1186
madsr .....	1187
mandel .....	1190
mandol .....	1191
marimba .....	1193
massign .....	1196
max .....	1198
maxabs .....	1199
maxabsaccum .....	1200
maxaccum .....	1201
maxalloc .....	1202
max_k .....	1204
mclock .....	1205
mdelay .....	1206
metro .....	1208
midic14 .....	1210
midic21 .....	1212
midic7 .....	1214
midichannelaftertouch .....	1216
midichn .....	1218
midicontrolchange .....	1221
midictrl .....	1223
mididefault .....	1224
midiiin .....	1225
midinoteoff .....	1228
midinoteoncps .....	1230
midinoteonkey .....	1232
midinoteonoct .....	1234

midinoteonpch .....	1236
midion .....	1238
midion2 .....	1241
midout .....	1242
midpitchbend .....	1244
midipolyaftertouch .....	1246
midiprogramchange .....	1248
miditempo .....	1249
midremot .....	1250
midglobal .....	1253
min .....	1254
minabs .....	1255
minabsaccum .....	1256
minaccum .....	1257
mirror .....	1258
MixerSetLevel .....	1259
MixerSetLevel_i .....	1261
MixerGetLevel .....	1262
MixerSend .....	1263
MixerReceive .....	1264
MixerClear .....	1266
mode .....	1267
modmatrix .....	1270
monitor .....	1275
moog .....	1276
moogladder .....	1278
moogvcf .....	1280
moogvcf2 .....	1282
moscil .....	1284
mp3in .....	1286
mpulse .....	1287
mrtmsg .....	1289
multitap .....	1290
mute .....	1291
mxadsr .....	1293
nchnls .....	1295
nestedap .....	1296
nlfilt .....	1299
noise .....	1301
noteoff .....	1304
noteon .....	1305
noteondur .....	1306
noteondur2 .....	1308
notnum .....	1310
nreverb .....	1312
nrpn .....	1315
nsamp .....	1316
nstrnum .....	1318
ntrpol .....	1319
octave .....	1320
octcps .....	1322
octmidi .....	1325
octmidib .....	1327
octmidinn .....	1329
octpch .....	1332
opcode .....	1335
OSCsend .....	1340
OSCinit .....	1342

OSClisten .....	1343
oscbnk .....	1347
oscil .....	1352
oscil1 .....	1354
oscil1i .....	1355
oscil3 .....	1356
oscili .....	1358
oscilikt .....	1360
osciliktp .....	1362
oscilikts .....	1364
osciln .....	1366
oscils .....	1367
oscilx .....	1369
out .....	1370
out32 .....	1371
outc .....	1372
outch .....	1373
outh .....	1374
outiat .....	1375
outic .....	1376
outic14 .....	1377
outipat .....	1379
outipb .....	1380
outipc .....	1381
outkat .....	1382
outkc .....	1383
outkc14 .....	1384
outkpat .....	1385
outkpb .....	1386
outkpc .....	1387
outleta .....	1390
outletk .....	1391
outletf .....	1392
outo .....	1393
outq .....	1394
outq1 .....	1395
outq2 .....	1396
outq3 .....	1397
outq4 .....	1398
outrg .....	1399
outs .....	1400
outs1 .....	1401
outs2 .....	1402
outvalue .....	1403
outx .....	1404
outz .....	1405
p .....	1406
p5gconnect .....	1408
p5gdata .....	1410
pan .....	1412
pan2 .....	1414
pareq .....	1415
partials .....	1418
partikkel .....	1420
partikkelsync .....	1429
passign .....	1430
pcauchy .....	1432
pchbend .....	1434

pchmidi .....	1436
pchmidib .....	1438
pchmidinn .....	1440
pchoct .....	1443
pconvolve .....	1446
pcount .....	1449
pdclip .....	1451
pdhalf .....	1454
pdhalfy .....	1457
peak .....	1460
peakk .....	1462
pgmassign .....	1463
phaser1 .....	1467
phaser2 .....	1470
phasor .....	1474
phasorbnk .....	1476
pindex .....	1478
pinkish .....	1480
pitch .....	1483
pitchamdf .....	1486
planet .....	1489
pluck .....	1491
poisson .....	1494
polyaft .....	1498
polynomial .....	1500
pop .....	1502
pop_f .....	1503
port .....	1504
portk .....	1505
poscil .....	1507
poscil3 .....	1510
pow .....	1513
powershape .....	1515
powoftwo .....	1517
prealloc .....	1519
prepiano .....	1521
print .....	1524
printf .....	1526
printk .....	1527
printk2 .....	1529
printks .....	1531
prints .....	1534
product .....	1536
pset .....	1537
ptrack .....	1538
puts .....	1540
push .....	1541
push_f .....	1542
pvadd .....	1543
pvbufread .....	1546
pvcross .....	1548
pvinterp .....	1550
pvoc .....	1552
pvread .....	1554
pvsadsyn .....	1556
pvsanal .....	1558
pvsarp .....	1561
pvsbandp .....	1564

pvsbandr .....	1566
pvsbin .....	1568
pvsblur .....	1570
pvsbuffer .....	1572
pvsbufread .....	1573
pvscale .....	1576
pvscent .....	1578
pvsccross .....	1580
pvsdemix .....	1582
pvsdiskin .....	1584
pvsdisp .....	1585
pvsfilter .....	1587
pvsfread .....	1590
pvsfreeze .....	1591
pvsftr .....	1593
pvsftw .....	1595
pvsfwrite .....	1597
pvshift .....	1599
pvsifd .....	1601
pvsinfo .....	1603
pvsinit .....	1604
pvsin .....	1605
pvsmaska .....	1606
pvsmix .....	1608
pvsmorph .....	1609
pvssmooth .....	1612
pvsout .....	1614
pvsosc .....	1615
pvspitch .....	1618
pvsstencil .....	1621
pvsvoc .....	1623
pvsynth .....	1625
pyassign Opcodes .....	1627
pycall Opcodes .....	1628
pyeval Opcodes .....	1632
pyexec Opcodes .....	1633
pyinit Opcodes .....	1636
pyrun Opcodes .....	1637
rand .....	1639
randh .....	1641
randi .....	1643
random .....	1645
randomh .....	1647
randomi .....	1649
rbjeq .....	1651
readclock .....	1654
readk .....	1656
readk2 .....	1658
readk3 .....	1660
readk4 .....	1662
reinit .....	1664
release .....	1666
remoteport .....	1667
remove .....	1668
repluck .....	1669
reson .....	1671
resonk .....	1673
resonr .....	1674



resonx .....	1677
resonxk .....	1679
resony .....	1680
resonz .....	1682
resyn .....	1684
reverb .....	1686
reverb2 .....	1688
reverb3c .....	1689
rewindscore .....	1691
rezzy .....	1692
rigoto .....	1694
rireturn .....	1695
rms .....	1697
rnd .....	1699
rnd31 .....	1701
round .....	1706
rspline .....	1707
rtclock .....	1708
s16b14 .....	1710
s32b14 .....	1712
scale .....	1714
samphold .....	1716
sandpaper .....	1717
scanhammer .....	1719
scans .....	1720
scantable .....	1722
scanu .....	1724
scoreline .....	1726
scoreline_i .....	1728
schedkwhen .....	1730
schedkwhennamed .....	1733
schedule .....	1735
schedwhen .....	1738
seed .....	1741
sekere .....	1742
semitone .....	1744
sense .....	1746
sensekey .....	1747
seqtime .....	1751
seqtime2 .....	1754
setctrl .....	1756
setksmps .....	1758
setscorepos .....	1760
sfilist .....	1761
sfinstr .....	1762
sfinstr3 .....	1764
sfinstr3m .....	1766
sfinstrm .....	1768
sfload .....	1770
sflooper .....	1771
sfpassign .....	1773
sfplay .....	1775
sfplay3 .....	1777
sfplay3m .....	1779
sfplaym .....	1781
sfplist .....	1783
sfpreset .....	1784
shaker .....	1786

sin .....	1788
sinh .....	1790
sininv .....	1792
sinsyn .....	1794
sleighbells .....	1796
slider16 .....	1798
slider16f .....	1800
slider32 .....	1802
slider32f .....	1804
slider64 .....	1806
slider64f .....	1808
slider8 .....	1810
slider8f .....	1812
slider16table .....	1814
slider16tablef .....	1816
slider32table .....	1818
slider32tablef .....	1820
slider64table .....	1822
slider64tablef .....	1824
slider8table .....	1826
slider8tablef .....	1828
sliderKawai .....	1830
sndload .....	1831
sndloop .....	1833
sndwarp .....	1835
sndwarpst .....	1839
socksend .....	1842
sockrecv .....	1844
soundin .....	1846
soundout .....	1849
soundouts .....	1851
space .....	1853
spat3d .....	1857
spat3di .....	1865
spat3dt .....	1869
spdist .....	1873
specaddm .....	1877
specdiff .....	1878
specdisp .....	1879
specfilt .....	1880
spechist .....	1881
specptrk .....	1882
specscal .....	1884
specsum .....	1885
spectrum .....	1886
splitrig .....	1888
spsend .....	1890
sprintf .....	1893
sprintfk .....	1894
sqrt .....	1896
sr .....	1898
stack .....	1899
statevar .....	1900
stix .....	1902
strchar .....	1904
strchark .....	1905
strcpy .....	1906
strcpyk .....	1907

strcat .....	1908
strcatk .....	1909
strcmp .....	1910
strcmpk .....	1911
streson .....	1912
strget .....	1914
strindex .....	1915
strindexk .....	1916
strlen .....	1917
strlenk .....	1918
strlower .....	1919
strlowerk .....	1920
strrindex .....	1921
strrindexk .....	1922
strset .....	1923
strsub .....	1925
strsubk .....	1927
strtod .....	1928
strtodk .....	1929
strtol .....	1930
strtolk .....	1931
strupper .....	1932
strupperk .....	1933
subinstr .....	1934
subinstrinit .....	1937
sum .....	1938
svfilter .....	1939
syncgrain .....	1942
syncloop .....	1944
syncphasor .....	1946
system .....	1950
tb .....	1952
tab .....	1955
tabrec .....	1957
table .....	1958
table3 .....	1960
tablecopy .....	1961
tablegpw .....	1962
tablei .....	1963
tableicopy .....	1964
tableigpw .....	1965
tableikt .....	1966
tableimix .....	1968
tableiw .....	1970
tablekt .....	1972
tablemix .....	1974
tableng .....	1976
tablera .....	1978
tableseg .....	1981
tablew .....	1982
tablewa .....	1985
tablewkt .....	1988
tablexkt .....	1991
tablexseg .....	1994
tabmorph .....	1995
tabmorpha .....	1997
tabmorphak .....	1999
tabmorphi .....	2001

tabplay .....	2003
tabsum .....	2004
tambourine .....	2005
tan .....	2007
tanh .....	2009
taninv .....	2011
taninv2 .....	2013
tbvcf .....	2015
tempest .....	2018
tempo .....	2021
tempoval .....	2023
tigoto .....	2025
timedseq .....	2026
timeinstk .....	2028
timeinsts .....	2030
timek .....	2032
times .....	2034
timeout .....	2036
tival .....	2037
tlineto .....	2038
tone .....	2039
tonek .....	2040
tonex .....	2041
trandom .....	2042
tradsyn .....	2043
transeg .....	2045
transegr .....	2047
trcross .....	2049
trfilter .....	2051
trhighest .....	2053
trigger .....	2054
trigseq .....	2056
trirand .....	2058
trlowest .....	2060
trmix .....	2061
trscale .....	2062
trshift .....	2063
trsplit .....	2064
turnoff .....	2066
turnoff2 .....	2068
turnon .....	2069
unirand .....	2070
upsamp .....	2072
urd .....	2073
vadd .....	2074
vadd_i .....	2077
vaddv .....	2079
vaddv_i .....	2082
vaget .....	2084
valpass .....	2086
vaset .....	2087
vbap16 .....	2089
vbap16move .....	2091
vbap4 .....	2093
vbap4move .....	2095
vbap8 .....	2097
vbap8move .....	2099
vbaplsinit .....	2102

vbapz .....	2104
vbapzmove .....	2106
vcella .....	2108
vco .....	2111
vco2 .....	2114
vco2ft .....	2118
vco2ift .....	2120
vco2init .....	2122
vcomb .....	2125
vcopy .....	2128
vcopy_i .....	2131
vdelay .....	2133
vdelay3 .....	2135
vdelayx .....	2137
vdelayxq .....	2139
vdelayxs .....	2141
vdelayxw .....	2143
vdelayxwq .....	2145
vdelayxws .....	2147
vdivv .....	2149
vdivv_i .....	2152
vdelayk .....	2154
vecdelay .....	2155
veloc .....	2156
vexp .....	2158
vexp_i .....	2161
vexpseg .....	2163
vexpv .....	2165
vexpv_i .....	2168
vibes .....	2170
vibr .....	2172
vibrato .....	2174
vincr .....	2177
vlimit .....	2178
vlinseg .....	2179
vlowres .....	2181
vmap .....	2183
vmirror .....	2185
vmult .....	2186
vmult_i .....	2190
vmultv .....	2192
vmultv_i .....	2195
voice .....	2197
vosim .....	2200
vphaseseg .....	2205
vport .....	2207
vpow .....	2208
vpow_i .....	2211
vpowv .....	2213
vpowv_i .....	2216
vpvoc .....	2218
vrandh .....	2220
vrandi .....	2223
vstaudio, vstaudiog .....	2226
vstbankload .....	2228
vstedit .....	2229
vstinit .....	2231
vstinfo .....	2233

vstmidiout .....	2235
vstnote .....	2237
vstparamset,vstparamget .....	2239
vstprogset .....	2241
vsubv .....	2242
vsubv_i .....	2245
vtable1k .....	2247
vtablei .....	2249
vtablek .....	2251
vtablea .....	2253
vtablewi .....	2255
vtablewk .....	2256
vtablewa .....	2258
vtabi .....	2260
vtabk .....	2262
vtaba .....	2264
vtabwi .....	2266
vtabwk .....	2267
vtabwa .....	2268
vwrap .....	2269
waveset .....	2270
weibull .....	2272
wgbow .....	2274
wgbowedbar .....	2276
wgbrass .....	2278
wgclar .....	2280
wgflute .....	2282
wgpluck .....	2284
wgpluck2 .....	2287
wguide1 .....	2289
wguide2 .....	2291
wiiconnect .....	2294
wiidata .....	2296
wiirange .....	2298
wiisend .....	2299
wrap .....	2300
wterrain .....	2301
xadsr .....	2303
xin .....	2305
xout .....	2307
xscanmap .....	2309
xscansmap .....	2310
xscans .....	2311
xscanu .....	2313
xtratim .....	2316
xyin .....	2319
zaci .....	2321
zakinit .....	2323
zamod .....	2326
zar .....	2328
zarg .....	2330
zaw .....	2332
zawm .....	2334
zfilter2 .....	2337
zir .....	2339
ziw .....	2341
ziwm .....	2343
zkci .....	2345

zkmod .....	2347
zkr .....	2349
zkw .....	2351
zkwm .....	2353
Instructions de Partition et Routines GEN .....	2356
Instructions de Partition .....	2356
Instruction a (ou Instruction Avancer) .....	2357
Instruction b .....	2358
Instruction e .....	2359
Instruction f (ou Instruction de Table de Fonction) .....	2360
Instruction i (Instruction d'Instrument ou de Note) .....	2363
Instruction m (Instruction de Marquage) .....	2367
Instruction n .....	2368
Instruction q .....	2370
Instruction r (Instruction Répéter) .....	2371
Instruction s .....	2373
Instruction t (Instruction de Tempo) .....	2374
Instruction v .....	2375
Instruction x .....	2377
Instruction { .....	2378
Instruction } .....	2381
Routines GEN .....	2381
GEN01 .....	2385
GEN02 .....	2388
GEN03 .....	2390
GEN04 .....	2392
GEN05 .....	2394
GEN06 .....	2396
GEN07 .....	2398
GEN08 .....	2400
GEN09 .....	2402
GEN10 .....	2405
GEN11 .....	2407
GEN12 .....	2409
GEN13 .....	2411
GEN14 .....	2414
GEN15 .....	2417
GEN16 .....	2418
GEN17 .....	2421
GEN18 .....	2422
GEN19 .....	2423
GEN20 .....	2425
GEN21 .....	2427
GEN22 .....	2429
GEN23 .....	2430
GEN24 .....	2431
GEN25 .....	2432
GEN27 .....	2433
GEN28 .....	2434
GEN30 .....	2436
GEN31 .....	2437
GEN32 .....	2438
GEN33 .....	2440
GEN34 .....	2442
GEN40 .....	2444
GEN41 .....	2445
GEN42 .....	2446
GEN43 .....	2447

GEN49 .....	2448
GEN51 .....	2450
GEN52 .....	2452
Les Programmes Utilitaires .....	2453
Répertoires. ....	2453
Formats des Fichiers Son. ....	2453
Génération d'un Fichier d'Analyse (ATSA, CVANAL, HETRO, LPANAL, PVANAL) .....	2454
Requêtes sur un Fichier (SNDINFO) .....	2465
Conversion de Fichier (, HET_EXPORT, HET_IMPORT, PVLOOK, PV_EXPORT, PV_IMPORT, SDIF2AD, SRCONV) .....	2467
Autres Utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MA- KECSD, MIXER, SCALE) .....	2482
Cscore .....	2495
Evénements, Listes et Opérations .....	2495
Ecrire un Programme de Contrôle Cscore .....	2498
Compiler un Programme Cscore .....	2503
Exemples Plus Avancés .....	2506
Etendre Csound .....	2508
Ajouter des Générateurs Unitaires .....	2508
Créer un Générateur Unitaire Intégré .....	2508
Ajouter un Générateur Unitaire comme Plugin .....	2512
Référence de OENTRY .....	2512
IV. Référence Rapide des Opcodes .....	2515
Référence Rapide des Opcodes .....	2517
A. Liste des exemples .....	2562
B. Conversion de Hauteur .....	2586
C. Valeurs d'Intensité du Son .....	2590
D. Valeurs de Formant .....	2591
E. Rapports de Fréquence Modale .....	2596
F. Fonctions Fenêtres .....	2598
G. Format de Fichier SoundFont2 .....	2603
H. Csound Double (64 bit) ou Float (32 bit) .....	2604
Glossaire .....	2606



---

# Préface

## Table des matières

Préface du Manuel de Csound .....	xxix
Histoire du Manuel de Référence Canonique de Csound .....	xxx
Mentions de copyright .....	xxxi
Débuter avec Csound .....	xxxiii
Les nouveautés de Csound 5.12 .....	xxxv

## Préface du Manuel de Csound

Barry Vercoe, MIT Media Lab

La réalisation de musique par ordinateur nécessite la synthèse de signaux audio avec des points discrets ou échantillons représentant des formes d'onde continues. Il y a de nombreuses façons de faire ceci, chacune offrant un type de contrôle différent. La synthèse directe génère des formes d'onde en échantillonnant une fonction enregistrée représentant une simple période ; la synthèse additive génère les nombreux partiels d'un son complexe, chacun ayant sa propre enveloppe d'intensité ; la synthèse soustractive démarre avec un son complexe pour le filtrer. La synthèse non-linéaire utilise la modulation de fréquence et la distorsion non-linéaire pour donner des caractéristiques complexes à des signaux simples, tandis que l'échantillonnage et l'enregistrement d'un son naturel permettent de l'utiliser à volonté.

Comme la spécification détaillée d'un son point par point est vite ennuyeuse, le contrôle est opéré de deux manières : 1) à partir d'instruments dans un orchestre, et 2) à partir d'événements dans une partition. Un orchestre est en fait un programme d'ordinateur qui peut produire des sons, tandis qu'une partition est un ensemble de données auxquelles ce programme réagit. Qu'une durée d'attaque soit une constante fixée dans un instrument, ou une variable de chaque note dans la partition, dépend de la façon dont l'utilisateur veut la contrôler.

Les instruments d'un orchestre de Csound (voir *Syntaxe de l'Orchestre*) sont définis dans une syntaxe simple qui invoque des procédures de traitement audio complexe. Une partition (voir *La Partition Numérique Standard*) passée à cet orchestre contient des informations de hauteur et de contrôle codées dans un format numérique standard. Bien que la plupart des utilisateurs se contentent de ce format, des langages de traitement de partition de plus haut niveau sont souvent pratiques.

Les programmes constituant le système Csound ont une longue histoire de développement, qui a commencé avec le programme Music 4 écrit aux Bell Telephone Laboratories au début des années 1960 par Max Mathews. C'est là que fut conçu le concept de table d'onde ainsi qu'une grande partie de la terminologie qui a permis depuis aux chercheurs de l'informatique musicale de communiquer. D'importantes additions furent apportées à Princeton par feu Godfrey Winham dans Music 4B ; mon propre Music 360 (1968) doit beaucoup à ce travail. Avec Music 11 (1973) j'ai pris une voie différente : les deux structures distinctes des signaux de contrôle et des signaux audio sont issues de mon engagement intensif lors des années précédentes dans la conception et l'élaboration de synthétiseurs numériques. Cette division a été retenue dans Csound.

Parce qu'il est entièrement écrit en C, on peut installer facilement Csound sur n'importe quelle machine équipée de Unix ou du langage C. Au MIT il tourne sur des stations VAX/DEC sous Ultrix 4.2, sur des machines SUN sous OS 4.1, sur SGI sous 5.0, sur IBM PC sous DOS 6.2 et Windows 3.1, et sur le Macintosh d'Apple sous ThinkC 5.0. Avec ce seul langage de définition de traitement numérique du signal et des formats audio portables comme AIFF et WAV, les utilisateurs peuvent passer facilement d'une machine à l'autre.

La version de 1991 apporta le vocodeur de phase, FOF, et les types de données spectrales. 1992 vit l'arrivée des convertisseurs et des unités de contrôle MIDI, permettant de piloter Csound depuis des fichiers MIDI (midifiles) et des claviers externes. En 1994 les programmes d'analyse du son (lpc, pvoc) furent intégrés dans le module principal, permettant de lancer tous les traitements de Csound depuis un seul exécutable, et Cscore pouvait passer les partitions directement à l'orchestre pour une réalisation itérative. La version de 1995 introduisit un ensemble MIDI étendu avec linseg basé sur MIDI, les filtres de Butterworth, la synthèse granulaire, et un détecteur de hauteur amélioré, dans le domaine fréquentiel. L'addition d'outils de génération d'événements en temps-réel (Cscore et MIDI) fut particulièrement importante, permettant des configurations excitation/réponse en temps-réel qui rendent possible la composition et l'expérimentation interactives. Il est apparu que la synthèse numérique par programme en temps-réel était désormais réellement prometteuse.

## Histoire du Manuel de Référence Canonique de Csound

La version initiale de ce manuel pour les premières versions de Csound fut démarrée au MIT par Barry L. Vercoe et y fut maintenue durant les années 1980 et le début des années 1990. Une partie du manuel provient de documents pour des programmes des années 1970 comme *Music11*. Ce manuel original fut amélioré et développé par Richard Boulanger, John ffitch, Jean Piché et Rasmus Ekman.

Ce manuel évolua vers le Manuel de Référence Officiel de Csound que l'on trouve toujours à <http://www.lakewoodsound.com/csound> [<http://www.lakewoodsound.com/csound/hypertext/manual.htm>], pour la version 4.16 de Csound, de novembre 1999, qui était maintenu par David M. Boothe.

Une version parallèle du manuel appelée le Manuel de Référence Alternatif de Csound, fut développée par Kevin Conder en utilisant *DocBook/SGML* [<http://www.docbook.org/>]. Cette version devint plus tard la version Canonique.

Quand le MIT plaça Csound sous license LGPL en 2003, le manuel passa sous license GFDL et fut placé sur Sourceforge avec les sources de Csound.

Durant l'hiver 2004, le Manuel Canonique fut converti en DocBook/XML par Steven Yi afin de permettre à plus de gens d'assurer sa compilation et sa maintenance.

Le manuel est actuellement maintenu par Andrés Cabrera avec des contributions continues de la communauté de Csound.

Le manuel est toujours un projet communautaire qui dépend des contributions des développeurs et des utilisateurs afin d'aider à affiner l'étendue et la précision de son contenu. Toutes les contributions sont les bienvenues et sont appréciées.

### Tableau 1. Autres Collaborateurs

Mike Berry
Eli Breder
Michael Casey
Michael Clark
Perry Cook
Sean Costello
Richard Dobson
Mark Dolson

Dan Ellis

Tom Erbe

Bill Gardner

Michael Gogins

Matt Ingalls

Richard Karpen

Anthony Kozar

Victor Lazzarini

Allan Lee

David Macintyre

Gabriel Maldonado

Max Mathews

Hans Mikelson

Peter Neubäcker

Peter Nix

Ville Pulkki

Maurizio Umberto Puxeddu

John Ramsdell

Marc Resibois

Rob Shaw

Paris Smaragdis

Greg Sullivan

Istvan Varga

Bill Verplank

Robin Whittle

Steven Yi

François Pinot

Cette liste n'est en aucune façon exhaustive. On peut obtenir plus d'information par le fichier Changelog dans l'entrepôt des sources du manuel.

## Mentions de copyright

Cette version du Manuel de Csound ("Le Manuel Canonique de Csound") est délivrée sous la GNU Free Documentation Licence [<http://www.gnu.org/licenses/fdl.txt>]. Les mentions de copyright antérieures et le crédit de leurs auteurs sont donnés ci-dessous, pour des raisons historiques.

## Mentions de copyright antérieures

Copyright © 1986, 1992 par le Massachusetts Institute of Technology. Tous droits réservés.

Développé par *Barry L. Vercoe* au Experimental Music Studio, Media Laboratory, M.I.T., Cambridge, Massachusetts, avec le support partiel de la System Development Foundation et du National Science Foundation Grant # IRI-8704665.

## Manuel

Copyright © 2003 by Kevin Conder pour les modifications apportées au Manuel de Référence Publique de Csound.

Il est permis de copier, distribuer et/ou modifier ce document selon les termes de la GNU Free Documentation License, Version 1.2 ou toute version ultérieure publiée par la Free Software Foundation ; sans aucune partie non modifiable, aucun texte de première de couverture et aucun texte de quatrième de couverture. Une copie de cette licence est disponible dans le sous-répertoire des exemples [exemples/fdl.txt] ou à : [www.gnu.org/licenses/fdl.txt](http://www.gnu.org/licenses/fdl.txt) [<http://www.gnu.org/licenses/fdl.txt>].

La documentation du langage Csound de ce manuel est dérivée du *Manuel de Référence Alternatif de Csound* de Kevin Conder, qui est lui-même dérivé du *Manuel de Référence Public de Csound*.

Copyright © 2004-2005 par Michael Gogins pour les modifications faites au *Manuel de Référence Alternatif de Csound*.

Cette mention légale provient du *Manuel de Référence Public de Csound* : « L'Edition Hypertexte originale du Manuel de Csound du MIT fut préparée pour le World Wide Web par *Peter J. Nix* du Department of Music at the University of Leeds et *Jean Piché* de la Faculté de musique de l'Université de Montréal. Une Edition d'Impression, en format Adobe Acrobat, fut ensuite maintenue par *David M. Boothe*. Les éditeurs reconnaissent entièrement les droits des auteurs de la documentation et des programmes originaux, comme décrits ci-dessus, et demandent en conséquence que cette mention soit citée chaque fois que ce matériel est utilisé. »

La dernière adresse réseau connue du Manuel de Référence Public de Csound était <http://www.lakewoodsound.com/csound/hypertext/manual.htm>.

L'adresse réseau du Manuel de Référence Alternatif de Csound, pour les copies Transparentes et les copies Opaques, est <http://kevindumpscore.com/download.html#csound-manual>.

L'adresse réseau du manuel de Csound et de CsoundAC est <http://sourceforge.net/projects/csound>.

Traduction française du manuel par François Pinot.

La traduction française du manuel est placée sous GNU Free Documentation License, Version 1.2 ou ultérieure, comme la version anglaise originale.

## Csound et CsoundAC

Csound est protégé par copyright de 1991 à 2008 par Barry Vercoe, John ffitch et les autres développeurs.

CsoundAC est protégé par copyright de 2001 à 2008 par Michael Gogins.

Csound et CsoundAC (anciennement CsoundVST) sont des logiciels libres ; vous pouvez les redistribuer et/ou les modifier selon les termes de la GNU Lesser General Public License tels que publiés par la Free Software Foundation ; soit la version 2.1 de la License, soit (à votre choix) n'importe quelle version ultérieure.

Csound et CsoundAC sont distribués dans l'espoir qu'il seront utiles, mais SANS AUCUNE GARANTIE ; sans même la garantie implicite de la VALEUR COMMERCIALE ou de l'ADEQUATION A UNE UTILISATION SPECIALE. Consultez la GNU Lesser General Public License pour plus de détails.

Vous devez avoir reçu une copie de la GNU Lesser General Public License en même temps que Csound et CsoundAC ; si ce n'est pas le cas, écrivez à la Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

# Virtual Synthesis Technology

Virtual Synthesis Technology (VST) PlugIn technologie d'interfaçage par Steinberg Soft- und Hardware GmbH.

## Débuter avec Csound

### Téléchargement

Si vous n'avez pas déjà installé Csound (ou si vous avez une ancienne version) téléchargez la version de Csound adaptée à votre plate-forme depuis la *Sourceforge Csound5 Download Page* [[http://sourceforge.net/project/showfiles.php?group\\_id=81968&package\\_id=120482](http://sourceforge.net/project/showfiles.php?group_id=81968&package_id=120482)]. Les installeurs pour Windows ont un suffixe '.exe' et ceux pour le Mac '.dmg' ou '.tar.gz'. Si le nom de l'installateur se termine en '-d' cela veut dire que l'installateur a été construit avec la *double* précision (64-bit) qui produit une sortie de meilleure qualité que la *simple* précision (32-bit), qui produit plus rapidement la sortie. Vous pouvez aussi télécharger les sources et les compiler, mais cela réclame plus d'expertise (voir la section *Construire Csound*).

Il est aussi utile de télécharger la version la plus récente de ce manuel, que vous trouverez également sur ce site.

### Exécution

Il y a différentes manières d'exécuter Csound. Comme Csound est un programme en ligne de commande (DOS dans la terminologie Windows), double-cliquer sur l'exécutable de Csound n'aura aucun effet. On doit appeler Csound soit depuis un terminal (ou invite DOS), soit depuis un frontal. Pour utiliser Csound en ligne de commande, vous devez ouvrir un *terminal* (une invite de commande DOS sous Windows). L'utilisation de Csound en ligne de commande pouvant sembler difficile si vous n'avez jamais utilisé de terminal, vous voudrez peut-être essayer un des frontaux inclus dans votre distribution. Un *frontal* est un programme graphique qui facilite l'exécution de Csound et peut souvent aider à éditer les fichiers csound.

Que ce soit avec un frontal ou en ligne de commande, l'exécution de Csound nécessite deux choses :

- Un fichier Csound ('.csd' ou bien un fichier '.orc' et un fichier '.sco')
- Une liste de drapeaux de ligne de commande (ou options de configuration) qui configurent l'exécution. Ils déterminent des éléments comme le nom et le format du fichier de sortie, si le temps-réel audio et le MIDI sont actifs, quelle carte son utiliser, la taille des tampons, la quantité de messages imprimés, etc. On peut inclure ces options dans le fichier '.csd' lui-même, aussi dans le cas des exemples inclus dans ce manuel, *vous ne devriez pas avoir en vous en soucier*. Vous pouvez trouver la liste complète et très longue des options de ligne de commande *ici*, mais vous voudrez peut-être la consulter plus tard...

Consultez la section *Configuration* si vous rencontrez des problèmes avec Csound.

Cette documentation comprend de nombreux fichiers '.csd' que vous pouvez tester, et qui devraient fonctionner directement depuis la ligne de commande ou depuis n'importe quel frontal. *oscil.csd* [exemples/oscil.csd] est un exemple simple que l'on peut trouver dans le répertoire des *exemples* de cette documentation. Votre frontal devrait vous permettre de choisir le fichier, et il devrait avoir un bouton 'play' ou 'render'.



### Note pour les utilisateurs de MacCsound

Il peut être nécessaire d'effacer toutes les lignes de la balise des options de commande afin

de faire fonctionner les exemples du manuel.

Vous pouvez aussi essayer les exemples à partir de la ligne de commande en vous déplaçant dans le répertoire des exemples du manuel avec ce type de commande sous Windows (en supposant que le manuel est situé en `c:\Program Files\Csound\manual\`) :

```
cd "c:\Program Files\Csound\manual\examples"
```

ou quelque chose comme :

```
cd /manualdirectory/manual/examples
```

pour les terminaux Mac ou linux et en tapant ensuite :

```
csound oscil.csd
```

Les fichiers exemples étant configurés pour fonctionner en temps-réel par défaut, vous devriez avoir entendu une onde sinusoïdale de 2 secondes.

## Ecrire vos propres fichiers .csd

Un fichier *.csd* ressemble à ceci (ce fichier est *oscils.csd* [examples/oscils.csd]) :

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o oscils.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a fast sine oscillator.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  a1 oscils iamp, icps, iphs
  out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Les fichiers *.csd* de Csound comprennent 3 sections principales entre les balises *<CsSynthesizer>* et *</CsSynthesizer>* :

- *CsOptions* - Contient les *options de ligne de commande* spécifiques à ce fichier particulier. Ces options peuvent aussi être définies dans le fichier *.csoundrc* ou directement dans la *ligne de commande*. Certains frontaux offrent également des moyens de spécifier les options globales ou locales.
- *CsInstruments* - Contient les instruments ou processus disponibles dans ce fichier. Les instruments sont définis en utilisant les codes d'opération *instr* et *endin*. La section *CsInstruments* contient aussi l'*En-tête de l'Orchestre* qui définit des choses comme le *taux d'échantillonnage*, le *nombre d'échantillons dans une période de contrôle* et le *nombre de canaux de sortie*.
- *CsScore* - Contient les 'notes' à jouer et optionnellement la définition de f-tables. Les notes sont créées en utilisant l'*instruction i*, et les f-tables sont créées en utilisant l'*instruction f*. Plusieurs autres *instructions de partition* sont disponibles.

Notez que tout ce qui suit un point-virgule (;) jusqu'à la fin de la ligne est un commentaire, et est ignoré par csound.

Vous pouvez écrire les fichiers csd dans n'importe quel éditeur de texte pur comme notepad ou textedit. Assurez vous de sauvegarder le fichier en texte pur (et non en texte enrichi). De nombreux *frontaux* proposent des capacités d'édition avancées avec coloration syntaxique et complétion.

Vous pouvez trouver ici [<http://csound.sourceforge.net/tutorial.pdf>] un tutoriel détaillé pour débiter avec Csound écrit par Michael Gogins.

## Les nouveautés de Csound 5.12

### Nouveautés dans la Version 5.12 (Janvier 2010)

- Nouveaux opcodes :
  - *transegr* est une version de l'opcode *transeg* qui a une section de relâchement déclenchée par midi, par un opcode *turnoff2* ou par un *événement de partition i* dont le numéro d'instrument est négatif.
  - *fgenonce* génère une table de fonction depuis la définition d'un instrument, sans duplication de données.
  - Les nouveaux *opcodes de Graphe de Fluence* permettent l'utilisation de graphes de fluences (graphes de flots de données asynchrones) dans Csound.
  - *passign* permet une initialisation rapide de variables de taux-i à partir de p-champs.
  - *crossfm* implémente la synthèse par modulation de fréquence croisée.
  - *loopxseg* est comme *loopseg* mais avec une enveloppe exponentielle.
  - *looptseg* est comme *loopseg* mais avec une enveloppe flexible comme *transeg*.
- Bogues corrigés et améliorations :
  - *pvshift* écrasait les données en mode double.
  - Le cas 3 de *pan2* a été fixé.
  - 
  - *clockon* et *clockoff* fonctionnent à nouveau.
  - *cross2* et *interp* pouvaient avoir des divisions par zéro.

- Le numéro de ligne dans les messages d'erreur n'inclut plus de texte de *.csoundrc*.
- *p5gconnect* a été modifié pour utiliser un processus léger séparé afin d'éviter les problèmes de temps mort.
- *transeg* vérifie le nombre de ses arguments.
- *sfload* se limitait à 10 soundfonts et n'était pas sécurisé. Il n'est plus limité.
- Changements Internes :
  - \" peut être utilisé comme caractère d'échappement dans les chaînes de l'orchestre.
  - Le nouveau parseur a été corrigé pour les arguments facultatifs.
  - Meilleure vérification de l'instruction *f* avec un nombre négatif.
  - Les soundfonts n'initialisent le tableau des hauteurs qu'une seule fois dans les opcodes de soundfont.
  - La collection usuelle de changements mineurs, de mises en forme et de commentaires.

## Nouveautés dans la Version 5.11 (Juin 2009)

- Nouveaux opcodes :
  - *mp3in* : permet la lecture des fichiers mp3 directement dans l'orchestre.
  - *wiiconnect*, *wiidata*, *wiisend*, *wiirange* opcodes par john ffitch pour recevoir et envoyer des données de ou vers un contrôleur wiimote.
  - Nouveaux opcodes pour recevoir des données directement d'un P5 Glove (gant de données) par john ffitch *p5gdata*
  - *tabsum* additionne des sections de ftables
  - *MixerSetLevel\_i* une version du taux d'initialisation seulement de *MixerSetLevel*
  - *doppler* implémente une simulation d'effet Doppler.
  - *filebit* retourne la profondeur binaire d'un fichier.
- Nouvelles fonctionnalités :
  - Nouveau type de panning pour l'opcode *pan2*
  - Nouvelle balise <CsExScore> de partition *csd*.
  - Nouvelle option -Ma pour le module ALSA RT MIDI qui écoute tous les périphérique.
  - Il y a un GEN49 pour lire des fichiers mp3.
  - Un code d'arrondi de bin a été ajouté à *pvscale*
  - Le support de taille de table non puissance de 2 a été ajouté à *ftload* et à *ftsave*



- GEN23 a été totalement réécrit pour être plus consistant dans ce qui constitue un séparateur et des commentaires. (Il n'y a toujours pas de commentaires /\* \*/)
- Bogues corrigés et améliorations :
  - Nouveaux exemples pour les opcodes pvs par by Joachim Heintz : pvsarp, pvscent, pvsbandp, pvsbandr, pvsbufread, pvsadsyn, pvsynth, pvsblur, pvscale, pvscross, pvsfilter, pvsfreeze, pvshift, pvsmaska, pvsmorph
  - L'utilisation de la numérotation automatique des ftables réutilise les numéros de table
  - seed avec un argument positif était défectueux
  - sprintf avec une chaîne vide imprimait des données erronées
  - mute fonctionne maintenant à la fois avec les instruments numérotés et nommés
  - Petites corrections dans diskinn et dans tablexkt
- Changements Internes :
  - SConstruct construit maintenant des bibliothèques partagées complètement indépendantes pour les adaptateurs de Python, Lua et Java.
  - Le nouveau Parseur est presque fonctionnel
  - Le retraçage des graphiques a été modifié de façon à ne redessiner que ceux qui sont sélectionnés.
  - Alsa-TR est plus tolérant sur les taux d'échantillonnage approchés.
  - Il est possible d'avoir une partition générée par un programme externe plutôt que d'utiliser le format de partition standard en spécifiant <CScore bin="translator"> pour appeler le programme de traduction des données de la partition.
  - lpc\_export corrigé.
  - La limite sur la longueur des noms de macro a été supprimée.
  - PMAX, le nombre d'arguments d'un événement de partition a été réduit de 2 unités, et un système de dépassement a été introduit pour que les GENs puissent avoir un nombre arbitraire d'arguments.
  - La version de l'API a été incrémentée à 2.1.
  - Nouvelle fonction de l'API ldmemfile2withCB() qui est une version de ldmemfile() qui permet de spécifier un callback appelé exactement une seule fois pour traiter le tampon MEMFIL après son chargement.
  - csound->floatsize corrigé; valait zéro dans les versions précédentes.
  - GetChannelLock ajouté.

## Nouveautés dans la Version 5.10 (Décembre 2008)

- Nouvelles fonctionnalités :

- Nouvelle option pour écouter tous les périphériques MIDI avec le module temps réel de portmidi. Pour activer l'écoute de tous les périphériques utiliser "-+rtmidi=portmidi -Ma".
- Implémentation du dithering sur la sortie ; le dithering rectangulaire et triangulaire est disponible dans certains cas.
- Le type 6 de *GEN20* a maintenant une option pour fixer la variance.
- Bogues corrigés et améliorations :
  - La variable d'environnement Locale a été fixée à "C numeric" pour éviter les problèmes de point décimal (, vs .).
  - Bogue corrigé dans *diskin*
  - *outo* était défectueux pour le canal 6
  - Bogue corrigé dans *pitchamdf*
  - L'initialisation de *zfilter2* a été corrigée
  - Bogue corrigé dans *s32b14*
  - D'autres bogues qui n'avaient pas été publiés ont été corrigés.
- Changements Internes :
  - La version majeure de l'API de Csound est incrémentée à 2, ceci affectant aussi csound.so. Ceci signifie que Csound 5.10 est incompatible avec les applications (frontaux, clients ou hôtes) construites pour Csound 5.08 et pour les versions antérieures qui utilisent la version 1.x de l'API. Il faudra reconstruire ces applications pour qu'elles fonctionnent avec les versions courante ou futures de Csound. Les frontaux pour Csound écrits dans des langages interprétés tels que Python ou Java pourront continuer à fonctionner sans modification. Il est également possible de garder une version antérieure de la bibliothèque de Csound et une version de l'API 2.0 sur la même machine afin que les applications basées sur l'ancienne ou sur la nouvelle version de Csound puissent coexister. Ces changements n'affectent en rien la compatibilité des orchestres et des partitions de Csound : tous les anciens documents devraient continuer à fonctionner comme par le passé.
  - Le temps est maintenant mesuré en interne en échantillons, ce qui résoud un ancien bogue concernant l'arrondi du temps au taux-k.
  - Plusieurs changements internes concernant les branchements. Certains opcodes sont significativement plus rapides.
- 

## Nouveautés dans la Version 5.09 (Octobre 2008)

- Nouveaux opcodes :
  - Nouvel opcode *vosim* par Rasmus Ekman qui recrée la technique historique VOSIM (VOcal SIMulator).
  - Nouvel opcode *dcblock2* par Victor Lazzarini.
  - Nouveau modèle de l'oscillateur de Chua : *chuap* par Michael Gogins.

- Nouveaux opcodes d'*Algèbre Linéaire* par Michael Gogins. Algèbre linéaire standard sur vecteurs et matrices réels et complexes : arithmétique composante à composante, normes, transposition et conjugaison, produits scalaires, matrice inverse, décomposition LU, décomposition QR et décomposition QR en valeurs propres. Inclut la copie de vecteur de et vers des signaux de taux-a, des tables de fonction et des signaux-f.
- Nouveaux opcodes ambisonic : *bformdec1* et *bformenc1*. Ces opcodes rendent obsolètes les anciens *bformdec* et *bformenc*.
- Nouveaux opcodes de contrôle de la partition par Victor Lazzarini : *rewindscoreet setscorepos*.
- Nouvelles fonctionnalités :
  - Les opcodes de la famille *vbap* (*vbap4*, *vbap8*, *vbap16* et *vbapz*) acceptent maintenant des variables de taux-k pour tous leurs arguments en entrée.
  - Nouveau module d'E/S pulseaudio sous Linux.
  - Nouveau paramètre facultatif *ienv* pour générer des enveloppes pour les opcodes de soundfont : *sfplay*, *sfplay3*, *sfplaym* et *sfplay3m*.
  - Ajout d'un 'argument de non-normalisation' à la routine GEN nommée "tanh". (Voir *Routines GEN Nommées*)
  - Ajout d'une option d'ordonnanceur de priorité sur alsa.
- Bogues corrigés et améliorations :
  - Notation scientifique permise dans *GEN23* (comme c'était le cas dans *csound4* !).
  - Bogue fixé dans l'initialisation de FLTK. L'utilisation de FLTK devrait être plus stable.
  - L'erreur sur les commentaires */\* \*/* dans l'orchestre a été corrigée.
  - *poscil* n'écrase plus la fréquence si la variable est partagée.
  - *printk* et *printks* vérifient que l'opcode est initialisé.
  - *soundout* et *soundouts* sont déclarés obsolètes en faveur de *fout*.
  - L'opcode *space* a été modifié pour accepter des tables dont la taille n'est pas une puissance de 2 (taille différée).
  - Un bogue de *pvsynth* a été corrigé.
- Changements Internes :
  - Le nouveau parseur reconnaît *#include* et les macros sans argument.
  - Moins de forçage de type entre floats et doubles dans la version float.
  - Un support expérimental multicore a été inclut.
  - L'opcode *buzz* a été réécrit.
  - Plusieurs autres changements internes et quelques corrections de bogues.

## Nouveautés dans la Version 5.08 (Février 2008)

- Nouveaux opcodes :
  - *imagecreate*, *imagesize*, *imagegetpixel*, *imagesetpixel*, *imagesave*, *imageload* et *imagefree*: nouveaux opcodes de traitement d'image par Cesare Marilungo pour lire/écrire des images png depuis Csound.
  - *pvsbandp* et *pvsbandr* par John ffitich, qui réalisent le filtrage passe-bande et réjection de bande dans le domaine spectral sur un signal pvs.
  - Nouveaux opcodes HRTF par Brian Carty : *hrtfmove*, *hrtfmove2* et *hrtfstat*.
  - Nouveaux opcodes de distorsion non-linéaire : *powershape*, *polynomial*, *chebyshevpoly*, *pdclip*, *pdhalf*, *pdhalfy*, et *syncphasor*
  - Nouvel opcode de contrôle de transport jack : *jacktransport*
- Nouvelles fonctionnalités :
  - Ajout de l'option de ligne de commande *--csd-line-nums=* pour sélectionner la façon de rapporter la ligne d'une erreur.
  - Nouvel opérateur de "non-report" (!) du langage de partition qui empêche le report implicite des p-champs dans les instructions i.
  - Ajout de l'option de ligne de commande *--syntax-check-only* (mutuellement exclusive avec *--i-only*)
  - Balise *<CsLicence>* pour les CSDs. *<CsLicense>* est acceptée comme une alternative à *<CsLicence>*.
- Bogues corrigés et améliorations :
  - L'ordre des sorties pour *hilbert* a été changé. Ce changement brise la compatibilité avec les versions précédentes, mais il fixe l'opcode qui travaille maintenant comme c'est décrit dans la documentation.
  - Les messages sur le chargement de plugins d'opcode ont été modifiés pour pouvoir être supprimés avec une option de niveau de message.
  - Changements majeurs sur les rapport d'erreur de partition ; les numéros de ligne dans la chaîne des entrées sont rapportés précisément pour la plupart des erreurs.
  - *pan2* a été corrigé afin d'être conforme à la documentation.
  - La balise *<CsVersion>* fonctionne à nouveau en conformité avec le manuel.
  - Les instructions de boucle { et } ont été fixées. Leur documentation ainsi que celles des opérateurs des expressions de partition ~, &, |, et # ont été ajoutées.
  - *hilbert* avait ses sorties permutées, c'est corrigé. L'exemple de manuel a été mis à jour.
- Changements Internes :
  - Changement de la localisation pour gettext ; les traductions en français et en espagnol (Colombie) sont disponibles.
  - Changements internes dans *partikkel*, interpolation de la lecture de forme d'onde et du fenêtrage, ce

qui permet une synthèse granulaire synchrone de hauteur plus précise. Exemples mis à jour pour *partikkel*.

- *pvscale* : algorithme amélioré pour la SDFT, supprimant la variation d'amplitude.

## Nouveautés dans la Version 5.07 (Octobre 2007)

- Nouveaux opcodes :
  - *pan2* : un opcode de spatialisation stéréo
  - *cpsmidinn*, *pchmidinn*, *octmidinn* : des convertisseurs de numéros de note MIDI
  - *fluidSetInterpMethod* : interpolation dans les SoundFonts de fluid
  - *sflooper* : une version SoundFont de *flooper2*
  - *pvsbuffer* et *pvsbufread* : mise en tampon/lecture de fsigs pour des changements de retards/échelle temporelle.
- Nouvelles fonctionnalités :
  - SDFT - la Transformée de Fourier Discrète à fenêtre Glissante -- intégrée aux opcodes *pvsanal*, etc si le recouvrement est inférieur à *ksmps* ou inférieur à 10. Certains opcodes *pvsXXX* sont étendus pour prendre des paramètres de taux-a dans cette situation.
  - Nouvelle option (*-O null* / *--logfile=null*) qui désactive tous les messages et toutes les impressions sur la console.
- Bogues corrigés et améliorations :
  - *partikkel* -- la synthèse par particule avait un bogue accidentel, corrigé.
  - La fermeture de l'entrée MIDI sur Windows(MM) échouait ; corrigé.
  - L'opcode *fluidEngine* prend maintenant comme paramètres facultatifs le nombre de canaux (compris entre 16 et 256) et le nombre de voix à jouer en polyphonie (compris entre 16 et 4096, la valeur par défaut étant 4096).
  - L'utilitaire *atsa* se comporte de manière plus sûre lorsqu'il reçoit du silence.
  - *ATSaddnz* : vérifications améliorées.
  - Les ambisonics (*bformdec*, *bformenc*) ont plus d'options pour le contrôle des opposés.
  - Le bogue dans *turnoff2* est corrigé.
  - *het\_export* : une vérification erronée plantait l'export.
- Changements Internes :
  - L'installateur sous Windows a été amélioré.
  - CsoundVST a été remplacé par CsoundAC, qui ne dépend pas des en-têtes du SDK de VST.

- Moins de messages lors du démarrage sous Windows(MM).
- Le type d'argument p (le taux-k vaut alors 1 par défaut) a été ajouté dans les types des paramètres d'entrée et de sortie des opcodes.

## Nouveautés dans la Version 5.06 (Juin 2007)

- Nouveaux opcodes granulaires : *partikkel*, *partikkelsync* et *diskgrain*.
- Nouvel opcode pour distribuer des évènements : *scoreline*.
- Plusieurs nouveaux opcodes en provenance de CsoundAV de Gabriel Maldonado : *hvs1*, *hvs2*, *hvs3*, *vphaseseg*, *inrg*, *outrg*, *lposcila*, *lposcilsa*, *lposcilsa2*, *tabmorph*, *tabmorpha*, *tabmorphi*, *tabmorphak*, *trandom*, *vtable1k*, *slider8table*, *slider16table*, *slider32table*, *slider64table*, *slider8tablef*, *slider16tablef*, *slider32tablef*, *slider64tablef*, *sliderKawai* et la version au taux-a de *ctrl7*.
- Egalement depuis CsoundAV, plusieurs nouveaux widgets FLTK : *FLkeyIn*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLmouse*, *FLxyin*, *FLhvsBox*, *FLslidBnkSet*, *FLslidBnkSetk*, *FLslidBnk2Set*, *FLslidBnk2Setk*, *FLslidBnkGetHandle*,
- De nouveaux opcodes pvs : *pvsdiskin*, *pvsmorph*,
- *eqfil*
- De nouvelles options de ligne de commande (*--m-warnings*) pour contrôler les messages
- *csladspa* : un kit de plugin CSD vers LADSPA.
- Et plusieurs corrections de bogues parmi lesquelles : version au taux-k de *system* ; problèmes de changement d'échelle de *vrandh* et de *vrandi* ; plantage occasionnel de *turnoff* ; bogue OS X ; *ATScross* et *mod*.

Csound5GUI fonctionne maintenant correctement sur toutes les plates-formes et csoundapi~ (objet pd) a été mis à jour.

---

# Partie I. Vue d'Ensemble

---

---

# Table des matières

Introduction .....	4
Développements Récents .....	5
Caractéristiques de Csound 5 .....	5
Caractéristiques de CsoundAC .....	6
La commande Csound .....	9
Ordre de priorité .....	9
Description de la syntaxe de la commande .....	9
Ligne de Commande de Csound .....	11
Options de Ligne de Commande (par Catégorie) .....	21
Variables d'Environnement de Csound .....	31
Format de Fichier Unifié pour les Orchestres et les Partitions .....	34
Description .....	34
Exemple .....	36
Fichier de Paramètres de Ligne de Commande (.csoundrc) .....	37
Prétraitement du Fichier Partition .....	37
La Fonction Extract .....	37
Prétraitement Indépendant avec Scsort .....	38
Utiliser Csound .....	39
Sortie Console de Csound .....	39
Comment Csound5 fonctionne .....	40
Valeurs d'amplitude dans Csound .....	41
Audio en temps-réel .....	43
Entrées/Sorties en temps-réel sur Linux .....	44
Windows .....	49
Mac .....	50
Optimisation de la Latence Audio en E/S .....	50
Configuration .....	52
Syntaxe de l'Orchestre .....	53
Instructions de l'En-tête de l'Orchestre .....	54
Instructions de Bloc d'Instrument et d'Opcode .....	54
Instructions Ordinaires .....	55
Types, Constantes et Variables .....	55
Initialisation de Variable .....	57
Expressions .....	57
Répertoires et Fichiers .....	57
Nomenclature .....	58
Macros .....	58
Instruments Nommés .....	59
Opcodes Définis par l'Utilisateur (UDO) .....	61
La Partition Numérique Standard .....	63
Prétraitement des Partitions Standard .....	63
Carry .....	63
Tempo .....	64
Sort .....	64
Instructions de Partition .....	65
Symboles Next-P et Previous-P .....	65
Ramping .....	66
Macros de Partition .....	67
Partition dans Plusieurs Fichiers .....	69
Evaluation des Expressions .....	70
Chaînes de caractères dans les p-champs .....	71
Frontaux .....	73
CsoundAC .....	74



CsoundVST .....	75
TclCsound .....	78
L'interpréteur Tcl : cstclsh .....	78
Cswish: le shell de fenêtrage .....	78
Un serveur Csound .....	79
Un Environnement de Scripting .....	80
TclCsound comme encapsuleur de langage .....	81
Référence des Commandes de TclCsound .....	81
Construire Csound .....	84
Liens Csound .....	90

---

# Introduction

Csound est un système de musique par ordinateur basé sur des générateurs unitaires et programmable par l'utilisateur. Il fut écrit à l'origine par Barry Vercoe au Massachusetts Institute of Technology en 1984 comme la première version en langage C de ce type de logiciel. Depuis, Csound a reçu de nombreuses contributions de la part de chercheurs, de programmeurs et de musiciens du monde entier.

Vers 1991, John ffitch porta Csound sur Microsoft DOS. De nos jours, Csound tourne sur plusieurs variétés de UNIX et de Linux, sur Microsoft DOS et Windows, sur toutes les versions du système d'exploitation du Macintosh y compris Mac OS X, et sur d'autres systèmes.

Il y a des systèmes de musique par ordinateur plus récents qui ont des éditeurs graphiques de patch (par exemple Max/MSP, PD, jMax, ou Open Sound World), ou qui utilisent des techniques d'ingénierie logicielle plus avancées (par exemple Nyquist ou SuperCollider). Cependant Csound possède toujours l'ensemble le plus important et le plus varié de générateurs unitaires, est le mieux documenté, s'exécute sur le plus grand nombre de plates-formes, et il est très facilement extensible. Il est possible de compiler Csound en utilisant l'arithmétique double précision pour obtenir une qualité sonore supérieure. Bref, on peut considérer Csound comme l'un des instruments de musique les plus puissants jamais créé.

En plus de cette version "canonique" de Csound et de CsoundAC, il existe d'autres versions de Csound et d'autres frontaux pour Csound, dont la plupart se trouvent sur <http://www.csounds.com>.

---

# Développements Récents

Depuis l'époque à laquelle Barry Vercoe écrivit la Préface originale de ce manuel, imprimée ci-dessus, de nombreuses nouvelles contributions ont été apportées à Csound. CsoundAC est une version étendue de Csound5.

## Caractéristiques de Csound 5

Csound 5 débute une nouvelle version majeure de Csound qui inclut les nouvelles caractéristiques suivantes :

- Autorisé maintenant sous la GNU Lesser General Public License, une licence de code source libre (open source).
- Un nouveau système de construction, plus facile à mettre en œuvre, utilisant SCons.
- L'utilisation de bibliothèques de code source libre largement acceptées :
  - libsndfile pour les entrées et les sorties dans les fichiers son.
  - PortAudio avec les pilotes ASIO pour les entrées et les sorties en temps-réel à faible latence.
  - FLTK pour les contrôles graphiques que l'on peut programmer dans le code de l'orchestre.
  - PortMidi pour les entrées et les sorties MIDI en temps-réel.

De plus, Istvan Varga a écrit des pilotes natifs MIDI et audio pour Windows et Linux.

- Un système simplifié de tampons audio.
- Des valeurs d'état retournées par toutes les fonctions internes, y compris les fonctions des opérateurs.
- Des opérateurs MIDI interopérables, ce qui permet d'utiliser les mêmes définitions d'instrument de façon interchangeable pour une exécution MIDI live ou une exécution différée commandée par une partition.
- Les opérateurs en plugin (module externe) sont opérationnels et sont acceptés plus largement. De nombreux opérateurs ont été déplacés dans des plugins. La plupart des nouveaux opérateurs sont des plugins, notamment :
  - Les opérateurs SoundFont basés sur FluidSynth.
  - Les opérateurs Python qui permettent d'exécuter du code Python dans l'en-tête d'un orchestre ou dans le code d'un instrument à cadence-*i* ou à cadence-*k*.
  - Les opérateurs Loris pour l'analyse temps/fréquence et la resynthèse.
  - Les opérateurs du bus de contrôle.
  - Les opérateur de mélangeur audio.
  - Les opérateurs de conversion de chaîne de caractères.

- Les opérateurs Open Sound Control (OSC) améliorés.
- Les opérateurs vectoriels.
- Les opérateurs pvs pour le traitement fréquentiel du signal en temps-réel, un portage du code du vocodeur de phase de Mark Dolson.
- Les opérateurs ATS pour l'analyse spectrale, la transformation, et la synthèse du son basée sur un modèle sinusoïdal avec bruit de bande critique. Un son dans ATS est un objet symbolique représentant un modèle spectral qu'on peut sculpter au moyen de diverses fonctions de transformation. Ces opérateurs peuvent lire, transformer et resynthétiser des fichiers d'analyse ATS. Il faut noter que l'application ATS est nécessaire pour produire les fichiers d'analyse.
- Les opérateurs STK, constitués par les instruments du Synthesis Toolkit original de Perry Cook en C++, adaptés en opérateurs.
- Les opérateurs d'adaptation DSSI et LADSPA pour accueillir des modules externes DSSI et LADSPA dans Csound.
- Les opérateurs d'adaptation vst4csVST pour accueillir des modules externes VST dans Csound. (Distribués seulement sous la forme de sources à cause des restrictions de la licence du SDK de VST.)
- Le fichier d'en-tête `OpcodeBase.hpp` pour écrire des modules externes en C++. C'est basé sur la technique du polymorphisme statique via l'héritage de template.
- le frontal `csound5gui` d'Istvan Varga pour Csound, qui simplifie l'édition de Csound et son utilisation spécialement pour les exécutions en direct, et le suivi de contrôle des exécutions.
- Les frontaux en Tcl/Tk de Victor Lazzarini pour Csound, `cstclsh` et `cswish`.
- L'API de Csound devient plus normalisée et est plus largement utilisée. Il existe des interfaces encapsulant l'API dans les langages suivants :
  - C (`include csound.h`).
  - C++ (`include csound.hpp`). Cette API contient les fonctions conteneur des fichiers de partition et d'orchestre de Csound.
  - Python (`import csnd`).
  - Java (`import csnd.*;`).
  - Lua (`require "csnd";`).
  - Lisp (utiliser le fichier CFFI `csound5.lisp`).
- Csound est maintenant totalement ré-entrant, ce qui veut dire que l'on peut exécuter plusieurs instances de Csound en même temps, dans le même processus.

John ffitich projette de remplacer l'analyseur syntaxique écrit à la main par un analyseur syntaxique produit à l'aide d'un générateur d'analyseur syntaxique, ce qui le rendrait moins sensible aux bogues et sans doute plus efficace.

## Caractéristiques de CsoundAC

CsoundAC est un module d'extension Python pour écrire de la musique en programmant en Python. CsoundAC est basé sur le concept de graphes de musique par Michael Gogins, dans lequel une partition est représentée par un arbre hiérarchisé de nœuds, qui peuvent contenir des notes, des générateurs de partition, des transformations de partition, et d'autres nœuds.

CsoundAC fournit aussi une interface Python vers l'API de Csound. Grâce à celle-ci, il est très facile d'utiliser Csound pour exécuter les compositions en CsoundAC. Avec les triples guillemets de Python, il est même possible d'inclure le code de l'orchestre de Csound pour une pièce directement dans le code Python de cette pièce, si bien que toute la programmation pour une pièce peut être maintenue dans un seul fichier.

Le système de coordonnées dans CsoundAC est basé sur un espace musical euclidien ayant pour dimensions {temps, durée, type d'évènement, numéro d'instrument, hauteur comme numéro de touche MIDI, intensité comme vélocité MIDI, phase, coordonnée spatiale X, coordonnée spatiale Y, coordonnée spatiale Z, ensemble de classes de hauteur, 1}. Un point dans cet espace musical peut être une note, une inflexion de note, ou même un grain sonore.

Un graphe de musique est un graphe orienté acyclique, ou un arbre, de nœuds dans l'espace musical. Ces nœuds sont associés avec des transformations locales du système de coordonnées. Il y a des nœuds pour contenir des partitions ou des fragments de partition, pour générer des partitions et pour transformer des partitions. De plus, chaque nœud peut contenir des nœuds enfants qui héritent du système de coordonnées du nœud parent.

Il est ainsi possible de composer une partition musicale en incluant ou en générant des notes dans les nœuds de niveau inférieur, puis en les assemblant dans une partition en utilisant des nœuds de niveau supérieur, et finalement en exécutant la partition avec Csound. Le procédé est strictement analogue à la construction d'une scène en 3 dimensions en synthèse graphique lorsque l'on génère des objets primitifs tels que sphères, cônes et cubes, puis qu'on les déplace dans l'espace pour assembler la scène.

Les classes de nœud dans CsoundAC sont :

- ScoreNode : contient simplement une séquence de notes ou d'autres points dans l'espace musical, peut-être importés d'un fichier MIDI.
- Rescale : met à l'échelle des points enfants pour les recadrer dans un intervalle donné de temps, durée, hauteur, et/ou d'autres dimensions.
- Cell : répète des points enfants en séquence à intervalles réguliers ; l'intervalle peut avoir une durée plus courte ou plus longue que celle des points enfants.
- Hocket : hoquet produit par des nœuds enfants.
- Lindenmayer : génère des partitions au moyen de systèmes de Lindenmayer O-L.
- StrangeAttractor : génère des partitions à partir de divers systèmes dynamiques chaotiques ajustables.
- MCRM : génère des partitions au moyen de l'algorithme de machine de copies multiples par réduction.
- ImageToScore : génère des partitions en transposant des fichiers image en points dans l'espace musical.
- Random : disperse des points enfants au hasard sur une ou plusieurs dimensions de l'espace musical, en utilisant diverses variables aléatoires.
- VoiceleadingNode : génère des progressions d'accords et des voix conductrices pour des notes enfants, au moyen d'opérations basées sur la théorie mathématique de la musique de Dmitri Tymoczko.

Enfin, le processus de composition peut inclure la dérivation d'une nouvelle classe Node en Python à partir d'un Node existant, afin de créer de nouveaux générateurs de partition et des transformations.

---

# La commande Csound

*Csound* est une commande pour générer une sortie son à partir d'un fichier *orchestre* et d'un fichier *partition* (ou d'un *fichier csd* unifié). Il a été conçu pour être appelé depuis un terminal ou une fenêtre DOS, mais on peut l'appeler depuis un *frontal* plus facile à utiliser. Le fichier partition peut être codé dans un des différents formats, au choix de l'utilisateur. La traduction, le tri et le formatage de la partition dans un texte numérique lisible par l'orchestre sont effectués par différents préprocesseurs ; tout ou partie de la partition est ensuite envoyé à l'orchestre. L'exécution de l'orchestre est influencée par des *options de commande*, qui fixent le niveau des comptes-rendus graphiques et de console, spécifient les noms des fichiers d'E/S et les formats d'échantillonnage, et déclarent la nature de la détection et du contrôle en temps-réel.

## Ordre de priorité

On peut fixer les options d'exécution de Csound en cinq endroits. Elles sont traitées dans l'ordre suivant :

1. Les valeurs par défaut de Csound
2. Le fichier défini par la *variable d'environnement* CSOUNDRC, ou le fichier .csoundrc dans le répertoire HOME
3. Le fichier .csoundrc dans le répertoire courant
4. La balise <CsOptions> dans un fichier .csd
5. En les passant sur la ligne de *commande* de Csound

Les dernières options dans la liste vont écraser les éventuelles options précédentes. A partir de la version 5.01 de Csound, les taux d'échantillonnage et de contrôle (options *-r* et *-k*) spécifiés n'importe où prévalent sur les valeurs sr, kr et ksmps définis dans l'en-tête de l'orchestre.

## Description de la syntaxe de la commande

La commande *csound* est suivie par un ensemble d'*Options de Ligne de Commande* et par les noms des fichiers de l'orchestre (.orc) et de la partition (.sco) ou du *Fichier Unifié csd* (contenant à la fois l'orchestre et la partition) à traiter. Les *Options de Ligne de Commande* pour contrôler la configuration d'entrée et de sortie peuvent apparaître n'importe où dans la ligne de commande, séparées ou collées ensemble. Un drapeau nécessitant un Nom ou un Nombre le trouvera dans l'argument lui-même ou dans celui qui le suit immédiatement. Les commandes suivantes sont donc équivalentes :

```
csound -nm3 nomorchestre -Sxxnomfichier nompartition  
csound -n -m 3 nomorchestre -x xnomfichier -S nompartition
```

Tous les drapeaux et les noms sont optionnels. Les valeurs par défaut sont :

```
csound -s -otest -b1024 -B1024 -m7 -P128 nomorchestre nompartition
```

où *nomorchestre* est un fichier contenant le code de l'orchestre Csound, et *nompartition* est un fichier de données de partition en format de partition numérique standard, facultativement pré-trié et réajusté en temps. Si *nompartition* est omis, il y a deux options par défaut :

1. si l'on attend une entrée en temps-réel (par exemple *-L*, *-M*, *-iadc* ou *-F*), un fichier partition factice est utilisé, constitué de la seule instruction 'f 0 3600' (c'est-à-dire écouter sur l'entrée TR pendant une heure)
2. sinon Csound utilise le dernier *score.srt* produit dans le répertoire courant.

Csound rend compte des différentes étapes de traitement de la partition et de l'orchestre lors de l'exécution, effectuant différents tests de syntaxe et d'erreurs. Une fois l'exécution commencée, les messages d'erreur proviennent soit du chargeur d'instrument soit des générateurs unitaires eux-mêmes. Une commande Csound peut inclure toute combinaison d'options bien formée.

## Exécuter les exemples du manuel à partir de la ligne de commande

La plupart des exemples du manuel sont prêts à l'emploi sans avoir besoin d'ajouter des options de ligne de commande, car ces options sont fixées dans la balise <CsOptions> du fichier csd, si bien qu'il suffit de taper une commande telle que :

```
csound oscil.csd
```

depuis le répertoire des exemples, et une sortie audio en temps-réel sera générée.



## Description

La commande *csound* exécute Csound.

## Syntaxe

```
csound [options] [nomorch] [nompartition]
```

```
csound [options] [nomfichiercsd]
```

## Options de la ligne de commande de Csound

Ci-dessous la liste par ordre alphabétique des options de ligne de commande disponibles dans Csound 5. Les implémentations sur différentes plates-formes peuvent ne pas réagir de la même façon à certaines options ! On peut consulter les options de ligne de commande par catégorie dans la section *Options de la Ligne de Commande (par Catégorie)*.

Les arguments de la ligne de commande sont de 2 types : arguments *options* (commençant par « - », « -- » ou « + »), et arguments *nom* (tels que noms de fichier). Certains arguments option sont suivis d'un nom ou d'un argument numérique. Les options qui commencent par « -- » et « + » prennent habituellement elles-mêmes un argument précédé du signe « = ».

### Options de la Ligne de Commande

-@ FICHIER	Une ligne de commande étendue est fournie par le fichier « FICHIER »
-3, --format=24bit	Utiliser des échantillons audio de 24 bit.
-8, --format=uchar	Utiliser des échantillons audio en caractères non-signés sur 8 bit.
--format=type	Choisir le format du fichier de sortie audio parmi les formats disponibles dans libsndfile. Actuellement la liste est aiff, au, avr, caf, flac, htk, ircam, mat4, mat5, nis, paf, pvf, raw, sd2, sds, svx, voc, w64, wav, wavex et xi. On peut aussi écrire --format=type:format ou --format=format:type pour fixer le type du fichier (wav, aiff, etc.) et le format d'échantillonnage (short, long, float, etc.) en même temps.
-A, --aiff, --format=aiff	Ecrire un fichier son au format AIFF. A utiliser avec les options -c, -s, -l, ou -f.
-a, --format=alaw	Utiliser des échantillons audio a-law.
-B NUM, - -hardwarebufsamps=NUM	Nombre de trames d'échantillonnage audio maintenues dans le tampon du <i>circuit</i> CNA. C'est une limite au-dessus de laquelle l'E/S audio <i>logicielle</i> va attendre avant de retourner. Une faible valeur réduit le délai audio d'E/S ; mais la valeur est souvent limitée par le matériel, et l'on risque des retards dans les données avec de petites valeurs. Dans le cas de la sortie portaudio (la sortie par défaut en temps-réel), le paramètre -B (plus précisément -B / sr) est passé comme valeur de "latence suggérée". En dehors de cela, Csound n'a aucun contrôle sur la manière dont PortAudio interprète le paramètre. La valeur par défaut est 1024 sur Linux, 4096

	sur Mac OS X et 16384 sur Windows.
-b NUM, --iobufsamps=NUM	<p>Nombre de trames d'échantillonnage audio dans chaque tampon <i>logiciel</i> d'E/S. De grandes valeurs conviennent, mais les petites valeurs réduiront le délai d'E/S audio et amélioreront la précision temporelle des événements en temps-réel. La valeur par défaut est 256 sur Linux, 1024 sur Mac OS X, et 4096 sur Windows. Lors d'une exécution en temps-réel, Csound attend les E/S audio toutes les <i>NUM</i> divisions. Il effectue aussi le traitement audio (et interroge d'autres entrées comme le MIDI) toutes les <i>ksmps</i> divisions de l'orchestre. On peut synchroniser les deux. Par commodité, si NUM est négatif, la valeur effective est <i>ksmps</i> * -NUM (audio synchrone avec les divisions de période k). Avec de petites valeurs de NUM (par exemple 1) l'interrogation devient fréquente et calée sur les divisions fixes d'échantillonnage du CNA.</p> <p>Note : si l'on utilise en même temps -iadc et -odac (audio temps-réel en mode duplex complet), il faut fixer l'option -b à un multiple entier de <i>ksmps</i>.</p>
-C, --cscore	Utiliser le traitement par Cscore du fichier partition.
-c, --format=schar	Utiliser des échantillons audio en caractères signés sur 8 bit.
--csd-line-nums=NUM	<p>Détermine comment les numéro de ligne sont comptés et affichés pour les messages d'erreur lors du traitement d'un fichier Csound Unified Document (.csd). Cette option n'a aucun effet si des fichiers d'orchestre et de partition séparés sont utilisés. (Csound 5.08 et versions ultérieures).</p> <ul style="list-style-type: none"> <li>• 0 = les numéros de ligne sont relatifs au début des sections de l'orchestre ou de la partition du CSD.</li> <li>• 1 = les numéros sont relatifs au début du fichier CSD. C'est le comportement par défaut dans Csound 5.08.</li> </ul>
-D, --defer-gen1	Différer le chargement des fichiers sons de GEN01 jusqu'au moment de l'exécution.
-d, --nodisplays	Supprimer tous les affichages. Voir -O si vous souhaitez enregistrer le compte-rendu dans un fichier.
--displays	Autoriser les affichages, inversant l'effet d'une éventuelle option -d précédente.
--default-paths	Autoriser à nouveau l'addition de répertoire de CSD/ORC/SCO aux chemins de recherche, si cette possibilité avait été désactivée par une option --no-default-paths précédente (par exemple dans .csoundrc).
--env:NOM=VALEUR	Positionner la variable d'environnement NOM à VALEUR. Note : on ne peut pas positionner toutes les variables d'environnement de cette manière, car certaines d'entre elles sont lues avant l'analyse de la ligne de commande. Cette option fonctionne avec INCDIR, SADIR, SFDIR, et SSDIR.
--env:NOM+=VALEUR	Ajouter VALEUR à la liste des chemins de recherche dont le séparateur est ';' dans la variable d'environnement NOM (ça peut-être INCDIR, SADIR, SFDIR, ou SSDIR). Si un fichier est trouvé

dans plusieurs répertoires, c'est le dernier qui est utilisé.

--expression-opt

*A partir de Csound 5.* Activer certaines optimisations dans les expressions :

- Les affectations redondantes sont éliminées chaque fois que c'est possible. Par exemple la ligne `a1 = a2 + a3` sera compilée en `a1 Add a2, a3` au lieu de `#a0 Add a2, a3 a1 = #a0` évitant une variable temporaire et un appel d'opcode. Moins d'appels d'opcode induisent une utilisation moindre du CPU (un orchestre moyen peut être compilé 10% plus vite avec --expression-opt, mais cela dépend aussi largement du nombre d'expressions utilisées, du taux de contrôle (voir également ci-dessous), etc ; ainsi, la différence peut être moindre, mais aussi beaucoup plus).
- le nombre de variables temporaires de taux a et de taux k est réduit significativement. L'expression

(a1 + a2 + a3 + a4)

sera compilée en

```
#a0 Add a1, a2
#a0 Add #a0, a3
#a0 Add #a0, a4          ; (le résultat se trouve dans #a0)
```

au lieu de

```
#a0 Add a1, a2
#a1 Add #a0, a3
#a2 Add #a1, a4          ; (le résultat se trouve dans #a2)
```

Les avantages d'avoir moins de variables temporaires sont :

- moins de mémoire cache utilisée, ce qui peut améliorer les performances des orchestres avec beaucoup d'expressions de taux a et un faible taux de contrôle (par exemple `ksmps = 100`)
- les grands orchestres sont chargés plus vite grâce au nombre moins important d'identifiants différents
- les erreurs de dépassement d'indice (par exemple quand des messages comme `Case2: indx=-56004 (ffff253c); (short)indx = 9532 (253c)` sont imprimés et que Csound a un comportement bizarre ou plante) peuvent être corrigées, car de telles erreurs sont provoquées par trop de noms de variable différents (spécialement au taux a) dans un seul instrument.

Noter que l'optimisation (pour des raisons techniques) n'est pas exécutée sur les i-variables temporaires.



## Avertissement

Lorsque --expression-opt est activé, il est interdit d'utiliser la fonction `i()` avec un argument expres-

sion, et il n'est pas prudent de compter au temps *i* sur la valeur de *k*-expressions.

-F FICHER, --midifile=FICHER	Lire les événements MIDI à partir du fichier <i>FICHER</i> . Le fichier ne doit avoir qu'une seule piste dans les versions 4.xx et antérieures de Csound ; cette limitation est levée à partir de Csound 5.00.
-f, --format=float	Utiliser des échantillons audio en format réel simple précision (non jouables sur certains systèmes, mais lisibles avec <i>-i</i> , <i>soundin</i> et <i>GENOI</i> ).
-G, --postscriptdisplay	Supprimer les graphiques, une sortie graphique PostScript est produite à la place.
-g, --asciidisplay	Supprimer les graphiques, une sortie pseudo-graphique ASCII est produite à la place.
-H#, --heartbeat=NUM	Imprimer un battement de cœur après chaque écriture de tampon dans le fichier son : <ul style="list-style-type: none"> <li>• pas de NUM, une barre tournante.</li> <li>• NUM = 1, une barre tournante.</li> <li>• NUM = 2, un point (.)</li> <li>• NUM = 3, la taille du fichier en secondes.</li> <li>• NUM = 4, un beep sonore.</li> </ul>
-h, --noheader	Pas d'en-tête dans le fichier son de sortie. N'écrit pas d'en-tête de fichier, seulement les échantillons binaires.
--help	Afficher un message d'aide en ligne.
-I, --i-only	<i>seulement au temps i</i> . Allouer et initialiser tous les instruments selon la partition, mais en ignorant tous les traitement de temps <i>p</i> (pas de <i>k</i> -signaux ni de <i>a</i> -signaux, et donc aucune amplitude et aucun son). Fournit un moyen rapide de tester la validité des <i>p</i> -champs de la partition et des <i>i</i> -variables de l'orchestre. Cette option est mutuellement exclusive avec l'option <i>--syntax-check-only</i> .
-i FICHER, --input=FICHER	Nom d'un fichier son en entrée. S'il ne s'agit pas d'un nom de chemin complet, le fichier sera d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement <i>SSDIR</i> (si elle définie), enfin par <i>SFDIR</i> . Si le nom est <i>stdin</i> , la lecture audio se fera à partir de l'entrée standard.  Les noms <i>devaudio</i> ou <i>adc</i> provoqueront l'écoute du son sur le périphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et 1023, ou un nom de périphérique séparé par un caractère : (par exemple <i>-iadc3</i> , <i>-iadc:hw:1,1</i> ). L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé pro-

voque habituellement une erreur et un affichage de la liste des numéros de périphérique valides.

Les données audio entrant grâce à *-i* peuvent être reçues au moyen d'opcodes tels que *inch*.

<code>--id_artist=chaîne</code>	(longueur max. = 200 caractères) Champ artiste dans le fichier son de sortie (pas d'espaces)
<code>--id_comment=chaîne</code>	(longueur max. = 200 caractères) Champ commentaire dans le fichier son de sortie (pas d'espaces)
<code>--id_copyright=chaîne</code>	(longueur max. = 200 caractères) Champ copyright dans le fichier son de sortie (pas d'espaces)
<code>--id_date=chaîne</code>	(longueur max. = 200 caractères) Champ date dans le fichier son de sortie (pas d'espaces)
<code>--id_software=chaîne</code>	(longueur max. = 200 caractères) Champ logiciel dans le fichier son de sortie (pas d'espaces)
<code>--id_title=chaîne</code>	(longueur max. = 200 caractères) Champ titre dans le fichier son de sortie (pas d'espaces)
<code>--ignore_csopts=entier</code>	S'il vaut 1, Csound ignorera toutes les options spécifiées dans la section CsOptions du fichier csd. Voir <i>Format de Fichier Unifié pour les Orchestres et les Partitions</i> .
<code>--input_stream=chaîne</code>	Nom du flot d'entrée pulseaudio.
<code>-J, --ircam, --format=ircam</code>	Ecrire un fichier son dans le format de l'IRCAM.
<code>--jack_client=[nom_client]</code>	Le nom de client utilisé par Csound, par défaut 'csound5'. Si plusieurs instances de Csound se connectent au serveur JACK, il faut utiliser différents noms de client pour éviter les conflits. (Linux et Mac OS X seulement)
<code>--jack_inportname=[préfixe du nom du port d'entrée], - --jack_outportname=[préfixe du nom du port de sortie]</code>	Préfixe du nom des ports JACK d'entrée/sortie de Csound ; la valeur par défaut est 'input' et 'output'. Le nom de port réel est le numéro de canal ajouté au préfixe du nom. (Linux et Mac OS X seulement)
	Exemple : avec les réglages par défaut ci-dessus, un orchestre stéréo créera ces ports dans une opération en full duplex :
	<pre>csound5:input1      (enregistrement gauche) csound5:input2      (enregistrement droite) csound5:output1     (reproduction gauche) csound5:output2     (reproduction droite)</pre>
<code>-K, --nopeaks</code>	Ne générer aucun bloc PEAK.
<code>-k NUM, --control-rate=NUM</code>	Remplacer le taux de contrôle ( <i>kr</i> ) fourni par l'orchestre.
<code>-L PERIPHERIQUE, - -score-in=PERIPHERIQUE</code>	Lire en temps-réel des événements de partition en ligne de texte à partir du périphérique <i>PERIPHERIQUE</i> . Le nom <i>stdin</i> permettra de recevoir les événements de partition de votre terminal, ou d'un autre processus via un tube de communication (pipe). Chaque ligne d'évènement est terminée par un retour chariot. Les évène-

ments sont codés de la même manière que ceux de la *partition numérique standard*, sauf qu'un évènement avec  $p2=0$  sera exécuté immédiatement, et qu'un évènement avec  $p2=T$  sera exécuté  $T$  secondes après son arrivée. Les évènements peuvent arriver n'importe quand et dans n'importe quel ordre. La fonction *carry* (*report de valeur*) de la partition est autorisée ici, ainsi que les notes liées ( $p3$  négatif) et les arguments chaîne, mais les pentes d'interpolation et les références *pp* ou *np* ne le sont pas.

-l, --format=long

Utiliser des échantillons audio codés en entiers longs.

-M PERIPHERIQUE, -  
-midi-device=PERIPHERIQUE

Lire les évènements MIDI à partir du périphérique *PERIPHERIQUE*. Si l'on utilise ALSA MIDI ( $-+rtmidi=alsa$ ), les périphériques sont sélectionnés par leur nom et pas par un numéro. Ainsi, il faut utiliser une option comme  $-M hw:CARTE,PERIPHERIQUE$  où *CARTE* et *PERIPHERIQUE* sont les numéros de la carte et du périphérique (par exemple  $-M hw:1,0$ ). Dans le cas de PortMidi et de MME, *PERIPHERIQUE* doit être un nombre, et s'il est en-dehors de l'intervalle permis, une erreur est levée et les numéros de périphérique valides sont imprimés. Avec PortMidi, on peut utiliser  $-Ma$  pour activer tous les périphériques. Cela marche aussi lorsqu'il n'y a pas de périphérique car aucune erreur n'est générée.

-m NUM, --messagelevel=NUM

Niveau des messages pour la sortie standard (terminal). Prend la *somme* de n'importe lesquelles de ces valeurs :

- 1 = messages d'amplitude de note
  - 2 = message d'échantillons hors intervalle
  - 4 = messages d'avertissement
  - 128 = impression d'information de tests de référence
- Et exactement un de ceux-ci pour choisir le format de l'amplitude des notes :
- 0 = amplitudes brutes, pas de couleur
  - 32 = dB, pas de couleur
  - 64 = dB, hors intervalle colorées en rouge
  - 96 = dB, toutes colorées
  - 256 = brutes, hors intervalle colorées en rouge
  - 512 = brutes, toutes colorées
- La valeur par défaut est 135 ( $128+4+2+1$ ), ce qui signifie tous les messages, valeurs d'amplitude brutes, et impression du temps écoulé à la fin de l'exécution. La mise en couleur des amplitudes brutes fut introduite dans la version 5.04.

--m-amps=NUM

Niveau des messages d'amplitudes sur la sortie standard (terminal).

- 0 = pas de messages d'amplitude de note
- 1 = messages d'amplitude de note

<code>--m-range=NUM</code>	<p>Niveau des messages de dépassement de limite sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = aucun message d'échantillon hors limites</li> <li>• 1 = messages d'échantillons hors limites</li> </ul>
<code>--m-warnings=NUM</code>	<p>Niveau des messages d'avertissement sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de messages d'avertissement</li> <li>• 1 = messages d'avertissement</li> </ul>
<code>--m-dB=NUM</code>	<p>Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = messages d'amplitude absolue</li> <li>• 1 = messages d'amplitude en dB</li> </ul>
<code>--m-colours=NUM</code>	<p>Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de coloration des messages d'amplitude</li> <li>• 1 = coloration des messages d'amplitude</li> </ul>
<code>--m-benchmarks=NUM</code>	<p>Niveau des messages d'information de test de performance sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de nombres de test de performance</li> <li>• 1 = nombres de test de performance</li> </ul>
<code>--max_str_len=entier</code>	<p>(min: 10, max: 10000) Longueur maximale des variables chaîne + 1 ; la valeur par défaut est 256 autorisant une longueur de 255 caractères. La longueur des constantes chaîne n'est pas limitée par ce paramètre.</p>
<code>--midi-key=N</code>	<p>Transmettre le numéro de touche d'un message MIDI note on au p-champ N en valeur MIDI [0-127].</p>
<code>--midi-key-cps=N</code>	<p>Transmettre le numéro de touche d'un message MIDI note on au p-champ N en cycles par seconde.</p>
<code>--midi-key-oct=N</code>	<p>Transmettre le numéro de touche d'un message MIDI note on au p-champ N en octave linéaire.</p>
<code>--midi-key-pch=N</code>	<p>Transmettre le numéro de touche d'un message MIDI note on au p-champ N en oct.pch (classe de hauteur).</p>
<code>--midi-velocity=N</code>	<p>Transmettre la vitesse d'un message MIDI note on au p-champ N en valeur MIDI [0-127].</p>
<code>--midi-velocity-amp=N</code>	<p>Transmettre la vitesse d'un message MIDI note on au p-champ N en amplitude [0-0dbfs].</p>
<code>--midioutfile=NOMFICHIER</code>	<p>Sauvegarder la sortie MIDI dans un fichier (seulement à partir de</p>

Csound 5.00).

<code>--msg_color=booléen</code>	Activer les attributs de message (couleurs etc.) ; il peut être nécessaire de les désactiver sur certains terminaux qui impriment des caractères étranges au lieu de modifier les attributs du texte. Par défaut : true.
<code>--mute_tracks=chaîne</code>	(longueur max. = 255 caractères) Ignorer les événements (autres que les changements de tempo) dans les pistes de fichier MIDI, définies par un motif binaire (par exemple, <code>--mute_tracks=00101</code> désactivera la troisième et la cinquième pistes).
<code>-N, --notify</code>	Avertir (par un beep) quand la partition ou la piste MIDI est terminée.
<code>-n, --nosound</code>	Pas de son. Faire tous les traitements, mais ne pas écrire de son sur le disque. Cette option ne change rien d'autre dans l'exécution.
<code>--no-default-paths</code>	Désactiver l'addition de répertoire de CSD/ORC/SCO au chemin de recherche.
<code>--no-expression-opt</code>	Désactiver l'optimisation des expressions.
<code>-O FICHER, --logfile=FICHER</code>	Compte-rendu dans le fichier <i>FICHER</i> . Si <i>FICHER</i> est null (c-à-d <code>-O null</code> ou <code>--logfile=null</code> ) toutes les impressions de message sur la console sont désactivées.
<code>-o FICHER, --output=FICHER</code>	Nom du fichier son de sortie. Si ce n'est pas un nom de chemin complet, le fichier son sera placé dans le répertoire donné par la variable d'environnement SFDIR (si elle est définie), sinon dans le répertoire courant. Le nom <i>stdout</i> provoque l'écriture audio sur la sortie standard, tandis qu'avec <i>null</i> il n'y a aucun son en sortie comme pour l'option <code>-n</code> . Si aucun nom n'est donné, le nom par défaut sera <i>test</i> .  Les noms <i>devaudio</i> ou <i>dac</i> (on peut utiliser <code>-odac</code> ou <code>-o dac</code> ) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : (par exemple <code>-odac3</code> , <code>-odac:hw:1,1</code> ). Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.
<code>--omacro:XXX=YYY</code>	Donner la valeur YYY à la macro d'orchestre XXX
<code>--output_stream=chaîne</code>	Nom du flot de sortie pulseaudio.
<code>-Q PERIPHERIQUE</code>	Activer les opérations MIDI OUT vers le périphérique d'id <i>PERIPHERIQUE</i> . Cette option permet l'exécution en parallèle sur MIDI OUT et CNA. Malheureusement le séquençement temps-réel implémenté dans Csound est complètement géré par le flot d'échantillons du tampon du CNA. C'est pourquoi les opérations MIDI OUT peuvent présenter quelques irrégularités dans le temps. On peut réduire ces irrégularités en utilisant une valeur plus faible pour l'option <code>-b</code> .

Si l'on utilise ALSA MIDI (`--rtmidi=alsa`), les périphériques sont



	sélectionnés par leur nom et non par un numéro. Il faut alors utiliser une option comme <code>-Q hw:CARTE,PERIPHERIQUE</code> où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple <code>-Q hw:1,0</code> ). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est hors intervalle, une erreur est levée et les numéros de périphérique valides sont imprimés.
<code>-R, --rewrite</code>	Réécrire continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).
<code>-r NUM, --sample-rate=NUM</code>	Remplacer le taux d'échantillonnage ( <i>sr</i> ) fourni par l'orchestre.
<code>++raw_controller_mode=booléen</code>	Désactiver le traitement spécial des contrôleurs MIDI tels que sustain, pédale, all notes off, etc., autorisant l'utilisation des 128 contrôleurs pour n'importe quelle fonction. Cela initialise également la valeur de tous les contrôleurs à zéro. Valeur par défaut : no.
<code>++rtaudio=chaîne</code>	(longueur max. = 20 caractères) Nom du module audio temps-réel. La valeur par défaut est PortAudio. Sont disponibles selon la plate-forme et les options de construction : Linux : alsa, jack; Windows : mme; Mac OS X : CoreAudio. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout plugin audio temps-réel.
<code>++rtmidi=chaîne</code>	(longueur max. = 20 caractères) Nom du module MIDI temps-réel. La valeur par défaut est PortMidi ; autres options (en fonction des options de construction) : Linux : alsa; Windows : mme, winmm. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout plugin MIDI temps-réel.  Les périphériques ALSA MIDI sont sélectionnés par leur nom au lieu d'un numéro. Aussi, il faut utiliser une option comme <code>-M hw:CARTE,PERIPHERIQUE</code> où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple <code>-M hw:1,0</code> ).
<code>-s, --format=short</code>	Utiliser des échantillons audio codés par des entiers courts.
<code>--sched</code>	<i>Seulement sur linux.</i> Utiliser pour le temps-réel le temps-partagé et le verrouillage de la mémoire. (nécessite également <code>-d</code> et <code>-o dac</code> ou <code>-o devaudio</code> ). Voir aussi <code>--sched=N</code> ci-dessous.
<code>--sched=N</code>	<i>Seulement sur linux.</i> Identique à <code>--sched</code> , mais permet de spécifier une valeur de priorité: si N est positif (dans l'intervalle 1 à 99) la politique de temps-partagé SCHED_RR sera utilisée avec une priorité de N ; autrement, SCHED_OTHER est utilisée avec le niveau "de gentillesse" (nice) à N. On peut aussi l'utiliser avec le format <code>--sched=N,MAXCPU,TEMPS</code> pour autoriser l'utilisation d'un processus léger (thread) de contrôle qui terminera Csound si le temps moyen d'utilisation de CPU dépasse MAXCPU pourcents sur une durée de TEMPS secondes (à partir de Csound 5.00).
<code>++server=chaîne</code>	Nom du serveur pulseaudio.
<code>++skip_seconds=float</code>	(min: 0) Commencer la reproduction au temps indiqué (en se-

	condes), en ignorant les évènements antérieurs de la partition ou du fichier MIDI.
--smacro:XXX=YYY	Donner la valeur YYY à la macro de partition XXX
--strset	<i>Csound 5.</i> L'option --strset permet de passer des chaînes à strset pour les lier à des valeurs numériques depuis la ligne de commande, dans le format '--strsetN=VALEUR'. Utile pour passer des paramètres à l'orchestre (par exemple des noms de fichier).
--syntax-check-only	Provoque l'arrêt de Csound immédiatement après que les parseurs de l'orchestre et de la partition ont fini la vérification de la syntaxe des fichiers d'entrée et avant que l'orchestre n'exécute la partition. Cette option est mutuellement exclusive avec l'option --i-only. (Csound 5.08 et versions ultérieures).
-T, --terminate-on-midi	Terminer l'exécution quand la fin du fichier MIDI est atteinte.
-t0, --keep-sorted-score	Empêcher Csound d'effacer le fichier de la partition triée, <i>score.srt</i> , lors de la sortie.
-t NUM, --tempo=NUM	Utiliser les pulsations non interprétées de <i>score.srt</i> pour cette exécution, et fixer le tempo initial à <i>NUM</i> pulsations par minute. Quand ce drapeau est positionné, le tempo de l'exécution de la partition est aussi contrôlable depuis l'orchestre. ATTENTION : ce mode d'opération est expérimental et n'est pas forcément fiable.
-U UTILITE, --utility=UTILITE	Invoker le programme utilitaire <i>UTILITE</i> . En donnant un nom invalide on obtient une liste des utilitaires.
-u, --format=ulaw	Utiliser des échantillons audio u-law.
-v, --verbose	Traduction et exécution détaillées. Imprime les détails de la traduction de l'orchestre et de son exécution, permettant une localisation plus précise des erreurs.
-W, --wave, --format=wave	Ecrire un fichier son au format WAV.
-x FICHIER, - -extract-score=FICHIER	Extraire un morceau de la partition triée, <i>score.srt</i> , en utilisant le fichier d'extraction <i>FICHIER</i> (voir <i>Extract</i> ).
-Z, --dither	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. La forme d'excitation par défaut est triangulaire.
-Z, --dither--triangular, - -dither--uniform	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. Dans le cas de -Z le chiffre qui suit doit être 1 (pour triangulaire) ou 2 (pour uniforme). L'interprétation exacte dépend du système de sortie.
-z NUM, --list-opcodesNUM	Lister les opcodes de cette version : <ul style="list-style-type: none"> <li>• pas de NUM, montrer seulement les noms</li> <li>• NUM = 0, montrer seulement les noms</li> <li>• NUM = 1, montrer les arguments de chaque opcode dans le format &lt;nomop&gt; &lt;argssortie&gt; &lt;argsentrée&gt;</li> </ul>

# Options de Ligne de Commande (par Catégorie)

Ci-dessous la liste par catégorie des options de ligne de commande disponibles dans Csound 5. Les implémentations sur différentes plates-formes peuvent ne pas réagir de la même façon à certaines options !

On peut consulter les options de ligne de commande par ordre alphabétique dans la section *Options de Ligne de Commande (par Ordre Alphabétique)*.

Le format d'une commande est soit :

```
csound [options] [nomorchestre] [nompartition]  
soit
```

```
csound [options] [nomfichiercsd]
```

où les arguments sont de 2 sortes : arguments *options* (commençant par « - », « -- » ou « + »), et arguments *nom* (tels que noms de fichier). Certains arguments options sont suivis d'un nom ou d'un argument numérique. Les options qui commencent par « -- » et « + » prennent habituellement un argument précédé du signe « = ».

## Sortie dans un Fichier Audio

-3, --format=24bit	Utiliser des échantillons audio de 24 bit.
-8, --format=uchar	Utiliser des échantillons audio en caractères non-signés sur 8 bit.
-A, --aiff, --format=aiff	Ecrire un fichier son au format AIFF. A utiliser avec les options -c, -s, -l, ou -f.
-a, --format=alaw	Utiliser des échantillons audio a-law.
-c, --format=schar	Utiliser des échantillons audio en caractères signés sur 8 bit.
-f, --format=float	Utiliser des échantillons audio en format réel simple précision (non jouables sur certains systèmes, mais lisibles avec -i, <i>soundin</i> et <i>GEN01</i> ).
--format=type	Choisir le format du fichier de sortie audio parmi les formats disponibles dans libsndfile. Actuellement la liste est aiff, au, avr, caf, flac, htk, ircam, mat4, mat5, nis, paf, pvf, raw, sd2, sds, svx, voc, w64, wav, wavex et xi. On peut aussi écrire --format=type:format ou --format=format:type pour fixer le type du fichier (wav, aiff, etc.) et le format d'échantillonnage (short, long, float, etc.) en même temps.
-h, --noheader	Pas d'en-tête dans le fichier son de sortie. N'écrit pas d'en-tête de fichier, seulement les échantillons binaires.
-i FICHIER, --input=FICHIER	Nom d'un fichier son en entrée. S'il ne s'agit pas d'un nom de chemin complet, le fichier sera d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement SSDIR (si elle définie), enfin par SFDIR. Si le nom est <i>stdin</i> , la lecture audio se fera à partir de l'entrée standard.

Les noms *devaudio* ou *adc* provoqueront l'écoute du son sur le pé-

riphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et 1023, ou un nom de périphérique séparé par un caractère : . L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé provoque habituellement une erreur et un affichage de la liste des numéros de périphérique valides.

Les données audio entrant grâce à *-i* peuvent être reçues au moyen d'opcodes tels que *inch*.

<i>-J, --ircam, --format=ircam</i>	Ecrire un fichier son dans le format de l'IRCAM.
<i>-K, --nopeaks</i>	Ne générer aucun bloc PEAK.
<i>-l, --format=long</i>	Utiliser des échantillons audio codés en entiers longs.
<i>-n, --nosound</i>	Pas de son. Faire tous les traitements, mais ne pas écrire de son sur le disque. Cette option ne change rien d'autre dans l'exécution.
<i>-o FICHIER, --output=FICHIER</i>	Nom du fichier son de sortie. Si ce n'est pas un nom de chemin complet, le fichier son sera placé dans le répertoire donné par la variable d'environnement SFDIR (si elle est définie), sinon dans le répertoire courant. Le nom <i>stdout</i> provoque l'écriture audio sur la sortie standard, tandis qu'avec <i>null</i> il n'y a aucun son en sortie comme pour l'option <i>-n</i> . Si aucun nom n'est donné, le nom par défaut sera <i>test</i> .  Les noms <i>dac</i> ou <i>devaudio</i> (on peut utiliser <i>-odac</i> ou <i>-o dac</i> ) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : . Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.
<i>-R, --rewrite</i>	Réécrire continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).
<i>-s, --format=short</i>	Utiliser des échantillons audio codés par des entiers courts.
<i>-u, --format=ulaw</i>	Utiliser des échantillons audio u-law.
<i>-W, --wave, --format=wave</i>	Ecrire un fichier son au format WAV.
<i>-Z, --dither</i>	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. La forme d'excitation par défaut est triangulaire.
<i>-Z, --dither--triangular, -dither--uniform</i>	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. Dans le cas de <i>-Z</i> le chiffre qui suit doit être 1 (pour triangulaire) ou 2 (pour uniforme). L'interprétation exacte dépend du système de sortie.

## Champs du Fichier de Sortie

<code>--id_artist=chaîne</code>	(longueur max. = 200 caractères) Champ artiste dans le fichier son de sortie (pas d'espaces)
<code>--id_comment=chaîne</code>	(longueur max. = 200 caractères) Champ commentaire dans le fichier son de sortie (pas d'espaces)
<code>--id_copyright=chaîne</code>	(longueur max. = 200 caractères) Champ copyright dans le fichier son de sortie (pas d'espaces)
<code>--id_date=chaîne</code>	(longueur max. = 200 caractères) Champ date dans le fichier son de sortie (pas d'espaces)
<code>--id_software=chaîne</code>	(longueur max. = 200 caractères) Champ logiciel dans le fichier son de sortie (pas d'espaces)
<code>--id_title=chaîne</code>	(longueur max. = 200 caractères) Champ titre dans le fichier son de sortie (pas d'espaces)

## Entrée/Sortie Audio en Temps-Réel

<code>-i adc[PERIPHERIQUE], -input=adc[PERIPHERIQUE]</code>	Les noms <i>devaudio</i> ou <i>adc</i> provoqueront l'écoute du son sur le périphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et 1023, ou un nom de périphérique séparé par un caractère : (par exemple <code>-iadc3</code> , <code>-iadc:hw:1,1</code> ). L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé provoque habituellement une erreur et un affichage de la liste des numéros de périphérique valides.
<code>-o dac[PERIPHERIQUE], -output=dac[PERIPHERIQUE]</code>	Les noms <i>dac</i> ou <i>devaudio</i> (on peut utiliser <code>-odac</code> ou <code>-o dac</code> ) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : (par exemple <code>-odac3</code> , <code>-odac:hw:1,1</code> ). Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.
<code>--rtaudio=chaîne</code>	(longueur max. = 20 caractères) Nom du module audio temps-réel. La valeur par défaut est PortAudio (sur toutes les plates-formes). Sont également disponibles selon la plate-forme et les options de construction : Linux : <code>alsa</code> , <code>jack</code> ; Windows : <code>mme</code> ; Mac OS X : <code>CoreAudio</code> . De plus, on peut utiliser <code>null</code> sur toutes les plates-formes, afin d'interdire l'utilisation de tout plugin audio temps-réel.
<code>--server=chaîne</code>	Nom du serveur pulseaudio.
<code>--output_stream=chaîne</code>	Nom du flot de sortie pulseaudio.
<code>--input_stream=chaîne</code>	Nom du flot d'entrée pulseaudio.
<code>--jack_client=[nom_client]</code>	Le nom de client utilisé par Csound, par défaut 'csound5'. Si plusieurs instances de Csound se connectent au serveur JACK, il faut

	utiliser différents noms de client pour éviter les conflits. (Linux et Mac OS X seulement)
--jack_inportname=[préfixe du nom du port d'entrée], - +jack_outportname=[préfixe du nom du port de sortie]	Préfixe du nom des ports JACK d'entrée/sortie de Csound ; la valeur par défaut est 'input' et 'output'. Le nom de port réel est le numéro de canal ajouté au préfixe du nom. (Linux et Mac OS X seulement)
	Exemple : avec les réglages par défaut ci-dessus, un orchestre stéréo créera ces ports dans une opération en full duplex :
	<pre>csound5:input1      (enregistrement gauche) csound5:input2      (enregistrement droite) csound5:output1     (reproduction gauche) csound5:output2     (reproduction droite)</pre>

## Entrée/Sortie par fichier MIDI

-F FICHIER, --midifile=FICHIER	Lire les événements MIDI à partir du fichier <i>FICHIER</i> . Le fichier ne doit avoir qu'une seule piste dans les versions 4.xx et antérieures de Csound ; cette limitation est levée à partir de Csound 5.00.
--midioutfile=NOMFICHIER	Sauvegarder la sortie MIDI dans un fichier (seulement à partir de Csound 5.00).
--mute_tracks=chaîne	(longueur max. = 255 caractères) Ignorer les événements (autres que les changements de tempo) dans les pistes de fichier MIDI, définies par un motif binaire (par exemple, --mute_tracks=00101 désactivera la troisième et la cinquième pistes).
--raw_controller_mode=booléen	Désactiver le traitement spécial des contrôleurs MIDI tels que pédale de sustain, all notes off, etc., autorisant l'utilisation des 128 contrôleurs pour n'importe quelle fonction. Cela initialise également la valeur de tous les contrôleurs à zéro. Valeur par défaut : no.
--skip_seconds=float	(min: 0) Commencer la reproduction au temps indiqué (en secondes), en ignorant les événements antérieurs de la partition ou du fichier MIDI.
-T, --terminate-on-midi	Terminer l'exécution quand la fin du fichier MIDI est atteinte.

## Entrée/Sortie MIDI en Temps-Réel

-M PERIPHERIQUE, - -midi-device=PERIPHERIQUE	Lire les événements MIDI à partir du périphérique <i>PERIPHERIQUE</i> . Si l'on utilise ALSA MIDI (--rtmidi=alsa), les périphériques sont sélectionnés par leur nom et pas par un numéro. Ainsi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est en-dehors de l'intervalle permis,
---	--

	une erreur est levée et les numéros de périphérique valides sont imprimés. Avec PortMidi, on peut utiliser '-Ma' pour activer tous les périphériques. Cela marche aussi lorsqu'il n'y a pas de périphérique car aucune erreur n'est générée.
--midi-key=N	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en valeur MIDI [0-127].
--midi-key-cps=N	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en cycles par seconde.
--midi-key-oct=N	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en octave linéaire.
--midi-key-pch=N	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en oct.pch (classe de hauteur).
--midi-velocity=N	Transmettre la vélocité d'un message MIDI note on au p-champ N en valeur MIDI [0-127].
--midi-velocity-amp=N	Transmettre la vélocité d'un message MIDI note on au p-champ N en amplitude [0-0dbfs].
--midioutfile=NOMFICHIER	Sauvegarder la sortie MIDI dans un fichier (seulement à partir de Csound 5.00).
++rtmidi=chaîne	(longueur max. = 20 caractères) Nom du module MIDI temps-réel. La valeur par défaut est PortMidi ; autres options (en fonction des options de construction) : Linux : alsa; Windows : mme, winmm. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout plugin MIDI temps-réel.  Les périphériques ALSA MIDI sont sélectionnés par leur nom au lieu d'un numéro. Aussi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0).
-Q PERIPHERIQUE	Activer les opérations MIDI OUT vers le périphérique d'id <i>PERIPHERIQUE</i> . Cette option permet l'exécution en parallèle sur MIDI OUT et CNA. Malheureusement le séquençement temps-réel implémenté dans Csound est complètement géré par le flot d'échantillons du tampon du CNA. C'est pourquoi les opérations MIDI OUT peuvent présenter quelques irrégularités dans le temps. On peut réduire ces irrégularités en utilisant une valeur plus faible pour l'option -b.  Si l'on utilise ALSA MIDI (++rtmidi=alsa), les périphériques sont sélectionnés par leur nom et non par un numéro. Il faut alors utiliser une option comme -Q hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -Q hw:1,0). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est hors intervalle, une erreur est levée et les numéros de périphérique valides sont imprimés.

## Affichage

<code>--csd-line-nums=NUM</code>	<p>Détermine comment les numéros de ligne sont comptés et affichés pour les messages d'erreur lors du traitement d'un fichier Csound Unified Document (.csd). Cette option n'a aucun effet si des fichiers d'orchestre et de partition séparés sont utilisés. (Csound 5.08 et versions ultérieures).</p> <ul style="list-style-type: none"><li>• 0 = les numéros de ligne sont relatifs au début des sections de l'orchestre ou de la partition du CSD.</li><li>• 1 = les numéros sont relatifs au début du fichier CSD. C'est le comportement par défaut dans Csound 5.08.</li></ul>
<code>-d, --nodisplays</code>	Supprimer tous les affichages. Voir <code>-O</code> si vous souhaitez enregistrer le compte-rendu dans un fichier.
<code>--displays</code>	Autoriser les affichages, inversant l'effet d'une éventuelle option <code>-d</code> précédente.
<code>-G, --postscriptdisplay</code>	Supprimer les graphiques, une sortie graphique PostScript est produite à la place.
<code>-g, --asciideisplay</code>	Supprimer les graphiques, une sortie pseudo-graphique ASCII étant produite à la place.
<code>-H#, --heartbeat=NUM</code>	<p>Imprimer un battement de cœur après chaque écriture de tampon dans le fichier son :</p> <ul style="list-style-type: none"><li>• pas de NUM, une barre tournante.</li><li>• NUM = 1, une barre tournante.</li><li>• NUM = 2, un point (.)</li><li>• NUM = 3, la taille du fichier en secondes.</li><li>• NUM = 4, un beep sonore.</li></ul>
<code>-m NUM, --messagelevel=NUM</code>	<p>Niveau des messages pour la sortie standard (terminal). Prend la <i>somme</i> de n'importe lesquelles de ces valeurs :</p> <ul style="list-style-type: none"><li>• 1 = messages d'amplitude de note</li><li>• 2 = message d'échantillons hors intervalle</li><li>• 4 = messages d'avertissement</li><li>• 128 = impression d'information de tests de référence</li></ul> <p>Et exactement un de ceux-ci pour choisir le format de l'amplitude des notes :</p> <ul style="list-style-type: none"><li>• 0 = amplitudes brutes, pas de couleur</li><li>• 32 = dB, pas de couleur</li><li>• 64 = dB, hors intervalle colorées en rouge</li><li>• 96 = dB, toutes colorées</li><li>• 256 = brutes, hors intervalle colorées en rouge</li></ul>



	<ul style="list-style-type: none"><li>• 512 = brutes, toutes colorées</li></ul> <p>La valeur par défaut est 135 (128+4+2+1), ce qui signifie tous les messages, valeurs d'amplitude brutes, et impression du temps écoulé à la fin de l'exécution. La coloration des amplitudes brutes fut introduite dans la version 5.04.</p>
--m-amps=NUM	<p>Niveau des messages d'amplitudes sur la sortie standard (terminal).</p> <ul style="list-style-type: none"><li>• 0 = pas de messages d'amplitude de note</li><li>• 1 = messages d'amplitude de note</li></ul>
--m-range=NUM	<p>Niveau des messages de dépassement de limite sur la sortie standard (terminal).</p> <ul style="list-style-type: none"><li>• 0 = aucun message d'échantillon hors limites</li><li>• 1 = messages d'échantillons hors limites</li></ul>
--m-warnings=NUM	<p>Niveau des messages d'avertissement sur la sortie standard (terminal).</p> <ul style="list-style-type: none"><li>• 0 = pas de messages d'avertissement</li><li>• 1 = messages d'avertissement</li></ul>
--m-dB=NUM	<p>Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).</p> <ul style="list-style-type: none"><li>• 0 = messages d'amplitude absolue</li><li>• 1 = messages d'amplitude en dB</li></ul>
--m-colours=NUM	<p>Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).</p> <ul style="list-style-type: none"><li>• 0 = pas de coloration des messages d'amplitude</li><li>• 1 = coloration des messages d'amplitude</li></ul>
--m-benchmarks=NUM	<p>Niveau des messages d'information de test de performance sur la sortie standard (terminal).</p> <ul style="list-style-type: none"><li>• 0 = pas de nombres de test de performance</li><li>• 1 = nombres de test de performance</li></ul>
++msg_color=booléen	<p>Activer les attributs de message (couleurs etc.) ; il peut être nécessaire de les désactiver sur certains terminaux qui impriment des caractères étranges au lieu de modifier les attributs du texte. Par défaut : true.</p>
-v, --verbose	<p>Traduction et exécution détaillées. Imprime les détails de la traduction de l'orchestre et de son exécution, permettant une localisation plus précise des erreurs.</p>
-z NUM, --list-opcodesNUM	<p>Lister les opcodes de cette version :</p>

- pas de NUM, montrer seulement les noms
- NUM = 0, montrer seulement les noms
- NUM = 1, montrer les arguments de chaque opcode dans le format <nomop> <argssortie> <argsentrée>

## Configuration et Contrôle de l'Exécution

-B NUM, -  
-hardwarebufsamps=NUM

Nombre de trames d'échantillonnage audio maintenues dans le tampon du *circuit* CNA. C'est une limite au-dessus de laquelle l'E/S audio *logicielle* va attendre avant de retourner. Une faible valeur réduit le délai audio d'E/S ; mais la valeur est souvent limitée par le matériel, et l'on risque des retards dans les données avec de petites valeurs. Dans le cas de la sortie portaudio (la sortie par défaut en temps-réel), le paramètre -B (plus précisément -B / sr) est passé comme valeur de "latence suggérée". En dehors de cela, Csound n'a aucun contrôle sur la manière dont PortAudio interprète le paramètre. La valeur par défaut est 1024 dans Linux, 4096 dans Mac OS X et 16384 dans Windows.

-b NUM, --iobufsamps=NUM

Nombre de trames d'échantillonnage audio dans chaque tampon *logiciel* d'E/S. De grandes valeurs fonctionnent bien, mais les petites valeurs réduiront le délai d'E/S audio et amélioreront la précision temporelle des événements en temps-réel. La valeur par défaut est 256 dans Linux, 1024 dans Mac OS X, et 4096 dans Windows. Lors d'une exécution en temps-réel, Csound attend les E/S audio toutes les NUM divisions. Il effectue aussi le traitement audio (et interroge d'autres entrées comme le MIDI) toutes les *ksmps* divisions de l'orchestre. On peut synchroniser les deux. Par commodité, si NUM est négatif, la valeur effective est *ksmps* \* -NUM (audio synchrone avec les divisions de période k). Avec de petites valeurs de NUM (par exemple 1) l'interrogation devient fréquente et calée sur les divisions fixes d'échantillonnage du CNA.

Note : si l'on utilise en même temps -iadc et -odac (audio temps-réel en mode duplex complet), il faut fixer l'option -b à un multiple entier de *ksmps*.

-k NUM, --control-rate=NUM

Remplacer le taux de contrôle (*kr*) fourni par l'orchestre.

-L PERIPHERIQUE, -  
-score-in=PERIPHERIQUE

Lire en temps-réel des événements de partition en ligne de texte à partir du périphérique *PERIPHERIQUE*. Le nom *stdin* permettra de recevoir les événements de partition de votre terminal, ou d'un autre processus via un tube de communication (pipe). Chaque ligne d'évènement est terminée par un retour chariot. Les événements sont codés de la même manière que ceux de la *partition numérique standard*, sauf qu'un événement avec p2=0 sera exécuté immédiatement, et qu'un événement avec p2=T sera exécuté T secondes après son arrivée. Les événements peuvent arriver n'importe quand et dans n'importe quel ordre. La fonction *carry* (*report de valeur*) de la partition est autorisée ici, ainsi que les notes liées (p3 négatif) et les arguments chaîne, mais les pentes d'interpolation et les références *pp* ou *np* ne le sont pas.

<code>--omacro:XXX=YYY</code>	Donner la valeur YYY à la macro d'orchestre XXX
<code>-r NUM, --sample-rate=NUM</code>	Remplacer le taux d'échantillonnage ( <i>sr</i> ) fourni par l'orchestre.
<code>--sched</code>	<i>Seulement dans linux.</i> Utiliser la planification du temps-réel et le verrouillage de la mémoire. (nécessite également <i>-d</i> et <i>-o dac</i> ou <i>-o devaudio</i> ). Voir aussi <code>--sched=N</code> ci-dessous.
<code>--sched=N</code>	<i>Seulement sur linux.</i> Identique à <code>--sched</code> , mais permet de spécifier une valeur de priorité: si N est positif (dans l'intervalle 1 à 99) la politique de planification SCHED_RR sera utilisée avec une priorité de N ; autrement, SCHED_OTHER est utilisée avec le niveau "de gentillesse" ( <i>nice</i> ) à N. On peut aussi l'utiliser avec le format <code>--sched=N,MAXCPU,TEMPS</code> pour autoriser l'utilisation d'un processus léger ( <i>thread</i> ) de surveillance qui terminera Csound si le temps moyen d'utilisation de CPU dépasse MAXCPU pourcents sur une durée de TEMPS secondes (à partir de Csound 5.00).
<code>--smacro:XXX=YYY</code>	Donner la valeur YYY à la macro de partition XXX
<code>--strset</code>	<i>Csound 5.</i> L'option <code>--strset</code> permet de passer des chaînes de caractères à <code>strset</code> depuis la ligne de commande, dans le format <code>'--strsetN=VALEUR'</code> . Utile pour passer des paramètres à l'orchestre (par exemple des noms de fichier).
<code>-+skip_seconds=float</code>	(min: 0) Commencer la reproduction au temps indiqué (en secondes), en ignorant les événements antérieurs de la partition ou du fichier MIDI.
<code>-t NUM, --tempo=NUM</code>	Utiliser les pulsations non interprétées de <i>score.srt</i> pour cette exécution, et fixer le tempo initial à NUM pulsations par minute. Quand cette options est positionnée, le tempo de l'exécution de la partition est également contrôlable depuis l'orchestre. ATTENTION : ce mode d'opération est expérimental et n'est pas forcément fiable.

## Divers

<code>-@ FICHIER</code>	Une ligne de commande étendue est fournie dans le fichier « FICHIER »
<code>-C, --cscore</code>	Utiliser le traitement par Cscore du fichier partition.
<code>--default-paths</code>	Autoriser à nouveau l'addition de répertoire de CSD/ORC/SCO aux chemins de recherche, si cette possibilité avait été désactivée par une option <code>--no-default-paths</code> précédente (par exemple dans <i>.csoundrc</i> ).
<code>-D, --defer-gen1</code>	Différer le chargement des fichiers sons de GEN01 jusqu'au moment de l'exécution.
<code>--env:NOM=VALEUR</code>	Positionner la variable d'environnement NOM à VALEUR. Note : on ne peut pas positionner toutes les variables d'environnement de cette manière, car certaines d'entre elles sont lues avant l'analyse de la ligne de commande. Cette option fonctionne avec INCDIR, SADIR, SFDIR, et SSDIR.

--env:NOM+=VALEUR

Ajouter VALEUR à la liste des chemins de recherche dont le séparateur est ';' dans la variable d'environnement NOM (ça peut-être INCDIR, SADIR, SFDIR, ou SSDIR). Si un fichier est trouvé dans plusieurs répertoires, c'est le dernier qui est utilisé.

--expression-opt

*A partir de Csound 5.* Activer certaines optimisations dans les expressions :

- Les affectations redondantes sont éliminées chaque fois que c'est possible. Par exemple la ligne `a1 = a2 + a3` sera compilée en `a1 Add a2, a3` au lieu de `#a0 Add a2, a3 a1 = #a0` évitant une variable temporaire et un appel d'opcode. Moins d'appels d'opcode induisent une utilisation moindre du CPU (un orchestre moyen peut être compilé 10% plus vite avec `--expression-opt`, mais cela dépend aussi largement du nombre d'expressions utilisées, du taux de contrôle (voir également ci-dessous), etc ; ainsi, la différence peut être moindre, mais aussi beaucoup plus).

- le nombre de variables temporaires de taux a et de taux k est réduit significativement. L'expression

```
(a1 + a2 + a3 + a4)
```

sera compilée en

```
#a0 Add a1, a2
#a0 Add #a0, a3
#a0 Add #a0, a4          ; (le résultat se trouve dans #a0)
```

au lieu de

```
#a0 Add a1, a2
#a1 Add #a0, a3
#a2 Add #a1, a4          ; (le résultat se trouve dans #a2)
```

Les avantages d'avoir moins de variables temporaires sont :

- moins de mémoire cache utilisée, ce qui peut améliorer les performances des orchestres avec beaucoup d'expressions de taux a et un faible taux de contrôle (par exemple `ksmps = 100`)
- les grands orchestres sont chargés plus vite grâce au nombre moins important d'identifiants différents
- les erreurs de dépassement d'indice (par exemple quand des messages comme `Case2: indx=-56004 (ffff253c); (short)indx = 9532 (253c)` sont imprimés et que Csound a un comportement bizarre ou plante) peuvent être corrigées, car de telles erreurs sont provoquées par trop de noms de variable différents (spécialement au taux a) dans un seul instrument.

Noter que l'optimisation (pour des raisons techniques) n'est pas exécutée sur les i-variables temporaires.



## Avertissement

Lorsque `--expression-opt` est activé, il est interdit d'utiliser la fonction `i()` avec un argument `expression`, et il n'est pas prudent de compter au temps `i` sur la valeur de `k-expressions`.

<code>--help</code>	Afficher un message d'aide en ligne.
<code>-I, --i-only</code>	<i>seulement au temps i</i> . Allouer et initialiser tous les instruments selon la partition, mais en ignorant tous les traitement de temps <code>p</code> (pas de <code>k-signaux</code> ni de <code>a-signaux</code> , et donc aucune amplitude et aucun son). Fournit un moyen rapide de tester la validité des <code>p-champs</code> de la partition et des <code>i-variables</code> de l'orchestre. Cette option est mutuellement exclusive avec l'option <code>--syntax-check-only</code> .
<code>--ignore_csopts=entier</code>	S'il vaut 1, Csound ignorera toutes les options spécifiées dans la section <code>CsOptions</code> du fichier <code>csd</code> . Voir <i>Format de Fichier Unifié pour les Orchestres et les Partitions</i> .
<code>--max_str_len=entier</code>	(min: 10, max: 10000) Longueur maximale des variables chaîne + 1 ; la valeur par défaut est 256 autorisant une longueur de 255 caractères. La longueur des constantes chaîne n'est pas limitée par ce paramètre.
<code>-N, --notify</code>	Avertir (par un beep) quand la partition ou la piste MIDI est terminée.
<code>--no-default-paths</code>	Désactiver l'addition de répertoire de <code>CSD/ORC/SCO</code> au chemin de recherche.
<code>--no-expression-opt</code>	Désactiver l'optimisation des expressions.
<code>-O FICHIER, --logfile=FICHIER</code>	Compte-rendu dans le fichier <i>FICHIER</i> . Si <i>FICHIER</i> est null (c-à-d. <code>-O null</code> ou <code>--logfile=null</code> ) toutes les impressions de message sur la console sont désactivées.
<code>--syntax-check-only</code>	Provoque l'arrêt de Csound immédiatement après que les parseurs de l'orchestre et de la partition ont fini la vérification de la syntaxe des fichiers d'entrée et avant que l'orchestre n'exécute la partition. Cette option est mutuellement exclusive avec l'option <code>--i-only</code> . (Csound 5.08 et versions ultérieures).
<code>-t0, --keep-sorted-score</code>	Empêche Csound d'effacer le fichier de la partition triée, <i>score.srt</i> , lors de la sortie.
<code>-U UTILITE, --utility=UTILITE</code>	Invoker le programme utilitaire <i>UTILITE</i> . En donnant un nom invalide on obtient une liste des utilitaires.
<code>-x FICHIER, -extract-score=FICHIER</code>	Extraire un morceau de la partition triée, <i>score.srt</i> , en utilisant le fichier d'extraction <i>FICHIER</i> (voir <i>Extract</i> ).

## Variables d'Environnement de Csound

Csound peut utiliser les variables d'environnement suivantes :

- **SFDIR** : Répertoire par défaut pour les fichiers son. Utilisé si aucun chemin complet n'est fourni pour les fichiers son.
- **SSDIR** : Répertoire par défaut pour les fichiers audio et MIDI en entrée (source). Utilisé si aucun chemin complet n'est fourni pour les fichiers son. On peut l'utiliser conjointement avec SFDIR pour fixer des répertoire d'entrée et de sortie séparés. Prière de noter qu'aussi bien les fichiers MIDI que les fichiers audio sont recherchés aussi dans SSDIR.
- **SADIR** : Répertoire par défaut pour les fichier d'analyse. Utilisé si aucun chemin complet n'est donné pour les fichiers d'analyse.
- **SFOUTYP** : Fixe le type par défaut du fichier de sortie. Actuellement ne sont valides que 'WAV', 'AIFF' et 'IRCAM'. Cette variable est testée par l'exécutable de csound et par les utilitaires et elle est utilisée si aucun type de fichier de sortie n'a été spécifié.
- **INCDIR** : Répertoire des fichiers à inclure. Spécifie l'endroit où se trouvent les fichiers utilisés par les instructions *#include*.
- **OPCODEDIR** : Définit l'endroit où se trouvent les plugins d'opcode en version simple précision (32 bit).
- **OPCODEDIR64** : Définit l'endroit où se trouvent les plugins d'opcode en version double précision (64 bit).
- **SNAPDIR** : Utilisée par les opcodes de contrôle graphique FLTK pour charger et sauvegarder les instantanés.
- **CSOUNDRC** : Définit le fichier de ressource (ou de configuration) de csound. Un chemin complet avec le nom d'un fichier contenant des options de csound doit être donné. Cette variable vaut *.csoundrc* par défaut.
- **CSSTRNGS** : A partir de Csound 5.00, la localisation des messages est contrôlée par les deux variables d'environnement CSSTRNGS et CS\_LANG, qui sont toutes deux optionnelles. CSSTRNGS pointe vers un répertoire contenant des fichiers *.xmg*.
- **CS\_LANG** : Sélectionne une langue pour les messages de csound.
- **RAWWAVE\_PATH** : Utilisée par les opcodes STK pour trouver les fichiers son bruts. Ne sert que si vous utilisez des opcodes de sur-couche STK comme STKBowed ou STKBrass.
- **CSNOSTOP** : Si cette variable d'environnement a pour valeur "yes", alors tous les affichages graphiques sont fermés à la fin de l'exécution (ce qui veut dire que vous n'en verrez peut-être pas grand chose dans le cas d'une exécution courte en temps différé). Dans le cas contraire, il faut cliquer sur "Quit" dans la fenêtre d'affichage FLTK pour sortir, ce qui permet de voir les graphiques même après que la fin de la partition soit atteinte.
- **MFDIR** : Répertoire par défaut pour les fichiers MIDI. Utilisé si aucun chemin complet n'est donné pour les fichiers MIDI. Prière de noter que les fichiers MIDI sont également recherchés dans SSDIR et SFDIR.

Pour plus d'information sur SFDIR, SSDIR, SADIR, MFDIR et INCDIR voir *Répertoires et Fichiers*.

Les seules variables d'environnement obligatoires sont OPCODEDIR et OPCODEDIR64. Il est très important de les remplir correctement, sinon la plupart des opcodes ne seront pas disponibles. Assurez-vous de fixer le chemin correctement en fonction de la précision de votre exécutable. Si vous lancez csound en ligne de commande sans aucun argument vous devriez voir un texte ressemblant à : Csound version 5.01.0 beta (float samples) Mar 23 2006. Ce texte fait référence à la version simple précision.

CSSTRNGS et CS\_LANG sont actuellement peu utiles car Csound n'a pas encore été complètement tra-

duit dans d'autres langues.

Voici d'autres variables d'environnement qui ne sont pas propres à Csound mais qui peuvent être importantes :

- *PATH* : Le répertoire contenant les exécutables de csound devrait être listé dans cette variable.
- *PYTHONPATH* : Si vous avez l'intention d'utiliser CsoundVST et python, le répertoire contenant la bibliothèque partagée \_CsoundVST et le fichier CsoundVST.py doit être dans votre variable d'environnement *PYTHONPATH* (ou le chemin de recherche par défaut de python), afin que l'interpréteur Python sache comment charger ces fichiers.
- *LADSPA\_PATH* et *DSSI\_PATH* : Ces variables d'environnement sont nécessaires si vous utilisez les opcodes du plugin *dssi4cs* (hôtes LADSPA et DSSI).
- *CSDOCDIR* : Spécifie le répertoire dans lequel se trouvent les fichiers d'aide html. Bien qu'elle ne soit pas utilisée directement par Csound, cette variable d'environnement peut aider les frontaux et les éditeurs (qui la mettent en œuvre) à trouver le manuel de csound.

## Fixer les variables d'environnement

### Sur la ligne de commande

On peut fixer les variables d'environnement sur la ligne de commande ou dans le fichier de configuration *.csoundrc* en utilisant l'option de ligne de commande `--env:NOM=VALEUR` ou `-env:NOM+=VALEUR`, où *NOM* est le nom de la variable d'environnement, et *VALEUR* est sa valeur. Voir *Options de Ligne de Commande*.



#### Note

Prière de noter que cette méthode ne fonctionnera pas pour les variables d'environnement qui sont lues avant les arguments de la ligne de commande. Pour *SADIR*, *SSDIR*, *SFDIR*, *INCDIR*, *SNAPDIR*, *RAWWAVE\_PATH*, *CSNOSTOP*, *SFOUTYP* cela devrait marcher, mais les variables d'environnement suivantes doivent être fixées dans le système avant de lancer csound : *OPCODEDIR*, *OPCODEDIR64*, *CSSTRINGS*, et *CS\_LANG*. Actuellement (v. 5.02) *CSOUNDRC* peut être fixée par `--env`, mais cette possibilité n'est pas garantie dans les versions futures.

## Windows

Pour fixer une variable d'environnement dans Windows XP et 2000 aller dans Panneau de Contrôle->Système->Avancé et cliquer sur le bouton 'Variables d'environnement'. Dans les autres versions de Windows antérieures à Windows XP et Windows 2000 on fixe les variables d'environnement dans le fichier *autoexec.bat*. Aller dans 'Poste de travail', choisir le lecteur C:, cliquer avec le bouton droit sur *autoexec.bat*, et choisir 'Edition'. Le format de l'instruction est : `SET NOM=VALEUR`.

## Linux

Il y a plusieurs manières de fixer les variables d'environnement sur linux. On peut les initialiser avec la commande de shell *export*, dans le fichier *.bashrc* ou des fichiers similaires ou en les ajoutant au fichier */etc/profile*.

## Mac

Si l'on a un Mac avec une version d'OS X inférieure à la 10.3 (y compris 10.2 et 10.1) il est alors possible que le shell par défaut soit le Tenex C-shell (tcsh). Si c'est le cas, il faut alors taper :

```
~% setenv OPCODEDIR "/Users/you/your/Csound5/build"
```

ou changer votre fichier /etc/profile et/ou modifier votre fichier .tcshrc.

Si l'on a un Mac avec OS X 10.3 ou 10.4 alors il y a certainement un shell C "Bourne-again" (bash) par défaut. Si c'est le cas, alors il faut taper quelque chose comme ça :

```
~$ export OPCODEDIR=/Users/you/your/Csound5/build
```

De plus si l'on a un bash shell par défaut, alors il est plus facile de modifier le fichier .bashrc ou le fichier /etc/profile.

A noter que si l'on choisit l'une des méthodes ci-dessus, par exemple modifier le fichier .bashrc, alors les variables d'environnement sont allouées quand un nouveau shell est créé. Ceci peut poser un problème lorsque votre application implémente une interface Quartz ou Aqua et n'utilise pas la ligne de commande.

Si c'est le cas, la solution standard (jusqu'à OS 10.3.9 et à moins que l'application utilise l'API de csound et fixe directement les variables d'environnement) consiste à créer un fichier contenant une liste de propriétés XML (un fichier nommé .plist par l'OS). Ce fichier devrait se trouver dans ~/.MacOSX/Environment.plist. Cette solution a été utilisée spécifiquement pour l'objet [csoundapi~] pour Pd sur OS X. Comme Pd utilise un style de paquetage .app natif OS X, et s'exécute en dehors de l'interface Aqua, les moyens standard de fournir les variables d'environnement à Csound ne fonctionnent pas. La solution consiste à fixer les variables d'environnement de Csound pour l'environnement Aqua.

Il est probable que la plupart des utilisateurs n'auront pas de répertoire caché .MacOSX dans leur répertoire \$HOME (alias ~/.). Il faut d'abord créer ce répertoire et y ajouter Environment.plist. Le contenu du fichier Environment.plist ressemblera à ceci :

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>OPCODEDIR</key>
<string>/Library/Frameworks/CsoundLib.framework/Versions/5.1/Resources/Opcodes</string>
<key>OPCODEDIR64</key>
<string>/Volumes/ExternalHD/devel/csound5/lib64</string>
<key>INCDIR</key>
<string>/Volumes/ExternalHD/CSOUND/include</string>
<key>SFDIR</key>
<string>/Volumes/ExternalHD/iTunes/csoundaudio</string>
</dict>
</plist>
```

et ainsi de suite, en utilisant la balise XML <key> pour chaque variable d'environnement requise par l'API et la balise <string> pour le chemin correspondant dans le système.

Prière de noter qu'il faut se déconnecter et se reconnecter (login) pour que ces changements prennent effet.

## Format de Fichier Unifié pour les Orchestres et les Partitions

### Description



Le Format de Fichier Unifié, introduit à partir de la version 3.50 de Csound, permet de combiner dans le même fichier l'orchestre et la partition, ainsi que les options de ligne de commande. Le fichier a pour extension *.csd*. Ce format fut introduit à l'origine par Michael Gogins dans AXCsound.

Le fichier est un fichier de données structurées qui utilise un langage de balises, de la famille SGML comme HTML. Une balise ouvrante (*<balise>*) et une balise fermante (*</balise>*) servent à délimiter les différents éléments. Ce fichier est sauvegardé comme un fichier texte.

## Format du Fichier de Données Structurées

### Éléments Obligatoires

la première balise du fichier doit être la balise ouvrante *<CsoundSynthesizer>*. La dernière balise du fichier doit être la balise fermante *</CsoundSynthesizer>*. Cet élément sert à avertir le compilateur csound du format *.csd*. Tout texte situé avant la balise de début et après la balise de fin est ignoré par Csound. Cette balise peut aussi s'écrire *<CsoundSynthesiser>*.

### Options (*<CsOptions>*)

Les *options de ligne de commande* de Csound sont insérées dans l'Élément Options. La section est délimitée par la balise ouvrante *<CsOptions>* et par la balise fermante *</CsOptions>*. Les lignes commençant par # ou par ; sont traitées comme des commentaires.

### Orchestre (*<CsInstruments>*)

Les définitions d'instruments (orchestre) sont mises dans l'Élément Instruments. Les instructions et la syntaxe de cette section sont identiques à celles du *fichier orchestre* de Csound, et répondent aux mêmes besoins, y compris les instructions d'en-tête (*sr*, *kr*, etc). Cet Élément Instruments est délimité par la balise ouvrante *<CsInstruments>* et par la balise fermante *</CsInstruments>*.

### Partition (*<CsScore>*)

Les instructions de la partition Csound sont mises dans l'Élément Score. Les instructions et la syntaxe de cette section sont identiques à celles du *fichier partition* de Csound, et répondent aux mêmes exigences. L'Élément Score est délimité par la balise ouvrante *<CsScore>* et par la balise fermante *</CsScore>*.

Les instructions de partition peuvent alternativement être générées par un programme externe en utilisant la balise *CsScore* avec un exécutable en attribut. Les lignes allant jusqu'à la balise fermante *</CsScore>* sont copiées dans un fichier et le programme externe nommé est appelé avec ce nom de fichier et le fichier de partition destinataire. Le programme externe doit créer une partition Csound standard.

### Éléments Optionnels

#### Inclusion de Fichiers Base64 (*<CsFileB>*)

On peut inclure des fichiers encodés en Base64 avec la balise *<CsFileB filename= nomfichier>*, où *nomfichier* est le nom du fichier à inclure. Les données encodées en Base64 doivent se terminer par une balise *</CsFileB>*. Pour encoder les fichiers, on peut se servir des utilitaires *csb64enc* et *makecsd* (inclus dans Csound à partir de la version 5.00). Le fichier sera extrait dans le répertoire courant, et effacé à la fin de l'exécution. S'il existe déjà un fichier du même nom, il n'est pas écrasé, mais au contraire, une erreur est levée.

On peut inclure des fichiers MIDI encodés en Base64 avec la balise *<CsMidifileB filename= nomfichier>*, où *nomfichier* est le nom du fichier qui contient l'information MIDI. Il n'y a pas de balise

fermante associée. Ceci a été ajouté dans la version 4.07 de Csound. Note : il n'est pas recommandé d'utiliser cette balise ; il vaut mieux utiliser `<CsFileB>`.

On peut inclure des fichiers d'échantillons encodés en Base64 avec la balise `<CsSampleB filename=nomfichier>`, où *nomfichier* est le nom du fichier qui contient les échantillons. Il n'y a pas de balise fermante associée. Ceci a été ajouté dans la version 4.07 de Csound. Note : il n'est pas recommandé d'utiliser cette balise ; il vaut mieux utiliser `<CsFileB>`.

## Limitation de Version (<CsVersion>)

On peut se limiter à certaines versions de Csound en plaçant l'une de ces instructions entre la balise ouvrante `<CsVersion>` et la balise fermante `</CsVersion>` :

Before #.#

ou

After #.#

où #.# est le numéro de version de Csound requis. La deuxième instruction peut s'écrire simplement comme :

#.#

Ceci a été ajouté dans la version 4.09 de Csound.

## Information de Licence (<CsLicence> or <CsLicense>)

Des détails de licence peuvent être inclus entre la balise ouvrante `<CsLicence>` et la balise fermante `</CsLicence>`. Il n'y a pas de format pour cette information, n'importe quel texte est acceptable. Ce texte sera imprimé par Csound sur la console lorsque le CSD sera exécuté.

## Exemple

Ci-dessous un fichier exemple, test.csd, qui produit un fichier .wav échantillonné à 44,1 kHz contenant une seconde d'une onde sinus à 1 kHz. L'affichage est supprimé. test.csd a été créé à partir de deux fichiers, tone.orc et tone.sco, avec l'addition des options de ligne de commande.

```
<CsoundSynthesizer>;
; test.csd - un fichier Csound de données structurées

<CsOptions>
-W -d -o tone.wav
</CsOptions>

<CsVersion>      ; section facultative
Before 4.10      ; ces deux instructions testent si
After 4.08       ; la version de Csound est la 4.09
</CsVersion>

<CsInstruments>
; à l'origine, tone.orc
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
instr 1
a1 oscil p4, p5, 1 ; simple oscillateur
out a1
```

```
    endin
  </CsInstruments>

  <CsScore>
    ; à l'origine, tone.sco
    f1 0 8192 10 1
    i1 0 1 20000 1000 ; joue un son pur à un kHz pendant une seconde
    e
  </CsScore>

</CsoundSynthesizer>
```

## Fichier de Paramètres de Ligne de Commande (.csoundrc)

Si le fichier *.csoundrc* existe, il sera utilisé pour fixer les paramètres de la ligne de commande. Ceux-ci peuvent être redéfinis. Csound 5.00 et les versions ultérieures lisent ce fichier d'abord depuis le répertoire HOME (ou le chemin complet défini par la *variable d'environnement* CSOUNDRC), et ensuite depuis le répertoire courant. Si les deux existent, les options de *.csoundrc* du répertoire courant seront prioritaires. Ce fichier a la même forme qu'un fichier *.csd*, mais sans les balises. Les lignes commençant par # ou ; sont traitées comme des commentaires.

Un fichier *.csoundrc* peut contenir des éléments comme ceux-ci :

```
-+rtaudio=portaudio -odac2 -iadc2 -+rtmidi=winmme -M1 -Q1 -m0
```

Dans ce cas, csound générera sa sortie en temps réel et recevra son entrée en temps réel depuis le périphérique 2, en utilisant l'interface du pilote portaudio. Il utilisera les entrées et les sorties MIDI en temps réel sur l'interface 1. Il imprimera très peu de messages (-m0). Ces options seront utilisées par défaut en l'absence d'autres options spécifiées dans la balise <CsOptions> du fichier *.csd* ou dans la ligne de commande (voir *Ordre de priorité*).

## Prétraitement du Fichier Partition

### La Fonction Extract

Cette fonction va extraire une partie d'un fichier de partition numérique triée en suivant les instructions venant d'un fichier de contrôle. Le fichier de contrôle contient une liste d'instruments et deux points dans le temps depuis (from) et à (to), de la forme :

```
instruments 1 2 from 1:27.5 to 2:2
```

Les étiquettes des composants peuvent être abrégés en i, f et t. Les points dans le temps marquent le début et la fin de l'extraction en termes de :

```
[no de section] : [no de pulsation].
```

chacune des trois parties de l'argument est optionnelle. Les valeurs par défaut lorsque i, f ou t sont manquants sont :

tous les instruments, début de la partition, fin de la partition.

## Prétraitement Indépendant avec Scsort

Bien que le résultat de tout le prétraitement de la partition se trouvent dans le fichier *score.srt* après l'exécution de l'orchestre (il existe dès que le prétraitement de la partition est fini), l'utilisateur peut vouloir parfois lancer ces phases indépendamment. La commande

```
scot nomfichier
```

va traiter le fichier au format Scot *nomfichier*, et produira comme résultat une *partition numérique standard* dans un fichier appelé *score* pour consultation ou traitement ultérieur.

La commande

```
scsort < fichierentrée > fichiersortie
```

effectuera les prétraitements de Report de Valeur (Carry), Tempo et Tri sur une partition numérique dans *fichierentrée*, déposant le résultat dans *fichiersortie*.

De même *extract*, lui aussi invoqué normalement comme élément de la *commande Csound*, peut être invoqué comme programme autonome :

```
extract xfile < partition.triée > extrait.partition
```

Cette commande attend une partition déjà triée. Une partition non triée doit d'abord passer par Scsort pour ensuite enchaîner avec le programme extract :

```
scsort < fichierpartition | extract xfile > extrait.partition
```

---

# Utiliser Csound

On peut faire fonctionner Csound dans divers modes et configurations. La méthode originale pour lancer Csound était un programme de console (invite DOS pour Windows, Terminal pour Mac OS X). Bien sûr, ceci fonctionne toujours. Lancer `csound` sans argument retourne une liste d'options de commande en ligne, qui sont expliquées plus en détail dans la section *Options de Ligne de Commande (par Catégorie)*. Normalement, l'utilisateur exécute quelque chose comme :

```
csound monfichier.csd
```

ou si l'on utilise des fichiers d'orchestre (`orc`) et de partition (`sco`) séparés :

```
csound monorchestre.orc mapartition.sco
```

On peut trouver plusieurs fichiers `.csd` dans le répertoire des exemples. La plupart des articles de ce manuel sur les opcodes incluent également des fichiers `.csd` simples montrant l'utilisation de l'opcode.

Il y a aussi plusieurs *Frontaux* que l'on peut utiliser pour lancer `csound`. Un *Frontal* est un programme graphique qui facilite la tâche de lancer `csound`, et qui fournit parfois des fonctionnalités d'édition et de composition.

Csound a aussi plusieurs moyens de produire une sortie. Il peut :

- Lire et écrire dans des fichiers son (restitution différée) - En utilisant les options `-o` et `-i` pour spécifier un fichier de sortie.
- Lire et écrire des données audio-numériques en utilisant une carte son (restitution en temps-réel) - En utilisant les options `-odac` et `-iadc`
- Lire et écrire dans des fichiers MIDI (temps différé) - En utilisant les options `-F` et `-midioutfile`.
- Lire et écrire des données MIDI en utilisant une interface et un contrôleur MIDI (contrôle en temps réel) - En utilisant les options `-M` et `-Q`.

## Sortie Console de Csound

Pendant son exécution, Csound écrit une sortie texte sur la console, qui montre des données sur cette exécution. Une sortie console ressemble à ceci :

```
time resolution is 0.455 ns
PortMIDI real time MIDI plugin for Csound
virtual_keyboard real time MIDI plugin for Csound
PortAudio real-time audio module for Csound
0dBFS level = 32768.0
Csound version 5.10 beta (float samples) Apr 19 2009
libsndfile-1.0.17
Reading options from $HOME/.csoundrc
UnifiedCSD: oscil.csd
STARTING FILE
Creating options
Creating orchestra
Creating score
orchname: /tmp/csound-XYACV6.orc
scorename: /tmp/csound-IYtLAJ.sco
rtaudio: ALSA module enabled
rtmidi: PortMIDI module enabled
orch compiler:
17 lines read
```

```

instr 1
Elapsed time at end of orchestra compile: real: 0.129s, CPU: 0.020s
sorting score ...
... done
Elapsed time at end of score sort: real: 0.130s, CPU: 0.020s
Csound version 5.10 beta (float samples) Apr 19 2009
displays suppressed
0dBFS level = 32768.0
orch now loaded
audio buffered in 256 sample-frame blocks
ALSA input: total buffer size: 1024, period size: 256
reading 1024-byte blks of shorts from adc (RAW)
ALSA output: total buffer size: 1024, period size: 256
writing 1024-byte blks of shorts to dac
SECTION 1:
ftable 1:
new alloc for instr 1:
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 10000.0 10000.0
Score finished in csoundPerform().
inactive allocs returned to freespace
end of score.          overall amps: 10000.0 10000.0
          overall samples out of range: 0 0
0 errors in performance
Elapsed time at end of performance: real: 2.341s, CPU: 0.050s
345 1024-byte soundblks of shorts written to dac
Removing temporary file /tmp/csound-CoVcrm.srt ...
Removing temporary file /tmp/csound-IYtLAJ.sco ...
Removing temporary file /tmp/csound-XYACV6.orc ...

```

La sortie console de Csound est assez fournie, particulièrement avant le début de l'exécution proprement dite (version, greffons chargés, etc.). L'exécution commence lorsqu'apparaît sur la console :

SECTION 1:

Dans cette exécution particulière, les lignes :

```

new alloc for instr 1:
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 10000.0 10000.0

```

montrent qu'une note pour l'instrument 1, durant 2 secondes et commençant à la date 0.000, a été produite avec une amplitude de 10000 sur les canaux 1 et 2. Une section importante de la sortie console est :

```

end of score.          overall amps: 10000.0 10000.0
          overall samples out of range: 0 0

```

qui montre l'amplitude globale et le nombre d'échantillons qui ont été rognés parce qu'ils étaient hors limites.

La ligne :

```

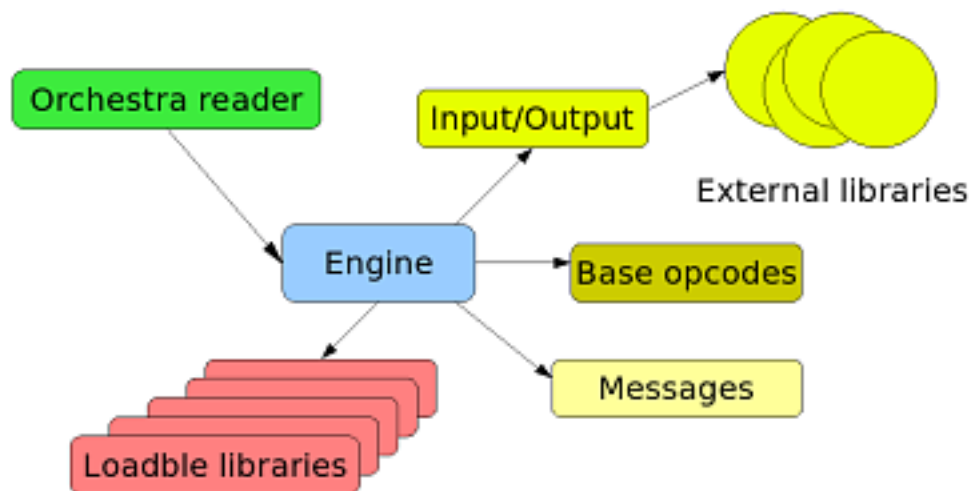
Elapsed time at end of performance: real: 2.341s, CPU: 0.050s

```

montre le temps d'horloge et le temps CPU utilisés par le processeur pour compléter le travail. Si le temps CPU est inférieur au temps d'horloge, cela veut dire que le csd peut être exécuté en temps réel (à moins qu'il ne contienne certaines sections très gourmandes en ressources CPU). La valeur "real time" est le temps total de traitement et il est supérieur car il comprend les accès disque, le chargement de modules, etc. (le temps CPU ne comptabilise que les calculs numériques). Si vous avez un son qui dure 100 secondes et que sa génération hors ligne ne dure que 5 secondes, cela veut dire que vous n'utilisez qu'environ 5% du CPU, et que son exécution ne nécessite que 0.05 du temps réel.

## Comment Csound5 fonctionne

Csound calcule et génère sa sortie en utilisant des "générateurs unitaires" (ugens) appelés *opcodes*. Ces opcodes sont utilisés pour définir des *instruments* dans l'*orchestre*. Quand vous lancez Csound, le moteur charge les Opcodes de base, et les opcodes contenus dans des "bibliothèques d'opcodes" séparées et chargeable. Il interprète ensuite l'orchestre (au moyen du chargeur d'orchestre). Le moteur met en place une chaîne de traitement des instruments, qui reçoit ensuite des événements depuis la partition ou en temps réel. La chaîne de traitement utilise les modules d'entrée/sortie pour générer la sortie. Il y a des modules qui peuvent écrire dans un fichier, ou générer une *sortie audio en temps réel*.



La Structure Modulaire de Csound5.

## Les tampons de traitement de Csound

Csound traite les données audio par blocs d'échantillons appelés tampons. Il y a trois couches de tampons séparées :

1. *spout* = tampon logiciel de bas niveau de Csound, contient *ksmps* trames d'échantillon. Csound traite les événements de contrôle en temps réel toutes les *ksmps* trames d'échantillon.
2. *-b* = Tampon logiciel intermédiaire de Csound (le tampon "logiciel"), en trames d'échantillon. Devrait être (mais ce n'est pas nécessaire) un multiple entier de *ksmps* (peut également être égal à *ksmps*). Une fois toutes les *ksmps* trames d'échantillon, Csound copie *spout* dans le tampon *-b*. Une fois toutes les *-b* trames d'échantillon, Csound copie le tampon *-b* dans le tampon "matériel" *-B*.
3. *-B* = tampon interne de la carte son (le tampon "matériel"), en trames d'échantillon. Devrait être (et cela peut être nécessaire) un multiple entier de *-b*. Si Csound n'arrive pas à délivrer un des *-b*, les trames d'échantillon *-b* en plus dans *-B* sont toujours là pour que la carte son continue de jouer tandis que Csound se rattrape. Mais ils peuvent être de la même taille si vous escomptez que Csound sera toujours en continuité avec la carte son.

## Valeurs d'amplitude dans Csound

Les valeurs d'amplitude dans Csound sont toujours relatives à une valeur "*0dbfs*" représentant l'amplitude de crête avant écrêtement, soit dans un codec AN/NA, soit dans un fichier son avec une étendue définie (ce qui est le cas de WAVE et de AIFF). A l'origine, dans Csound, cette valeur était toujours 32767, correspondant à l'étendue bipolaire d'un fichier son 16 bit ou d'un codec AN/NA 16 bit, les seules sorties possibles de Csound à l'époque. Ceci reste l'amplitude de crête *par défaut* dans Csound, pour une compatibilité descendante et vous verrez que la plupart des exemples de ce manuel utilisent toujours cette valeur (c'est pourquoi l'on trouve de grandes valeurs d'amplitude comme 10000).

La valeur *Odbfs* permet à Csound de produire des valeurs convenablement calibrées quelque soit le format utilisé, entiers sur 24 bit, nombres en virgule flottante sur 32 bit, ou même entiers sur 32 bit. Autrement dit, les valeurs d'amplitude littérales écrites dans un instrument de Csound ne concordent avec celles qui sont écrites *littéralement* dans le fichier que si la valeur *Odbfs* dans Csound correspond exactement à celle du format d'échantillonnage de la sortie. La conséquence de cette approche est que l'on peut écrire une pièce avec une certaine amplitude et en avoir une restitution correcte et identique (sans tenir compte bien sûr de la gamme dynamique meilleure des formats en haute résolution) qu'elle soit écrite dans un fichier de nombres entiers ou en virgule flottante, ou rendue en temps réel.



## Note

La seule exception à ceci se produit si l'on choisit d'écrire dans un format de fichier "brut" (sans en-tête). Dans de tels cas la valeur interne *Odbfs* est sans signification, et quelques soient les valeurs utilisées, elles sont écrites inchangées. Cela permet de faire générer ou traiter par Csound des données arbitraires. C'est une chose relativement exotique à faire, mais certains utilisateurs en ont besoin.

Vous pouvez choisir de redéfinir la valeur *Odbfs* dans l'en-tête de l'orchestre, par pure commodité ou selon vos préférences. Beaucoup de personnes choisiront 1,0 (le standard pour SAOL, d'autres logiciels comme Pure Date, et pour beaucoup de plugins standard comme VST, LADSPA, CoreAudio AudioUnits, etc), mais n'importe quelle valeur est possible.

Le facteur commun dans la définition des amplitudes est l'échelle en décibel (dB), avec  $0dB_{FS}$  toujours compris comme la crête numérique ; ainsi "0dbfs" veut dire valeur de "0dB Full-Scale" (sur l'étendue complète). Cette mesure est différentes des valeurs d'amplitude réelles, puisque celles-ci sont sur une échelle linéaire qui montre l'oscillation réelle autour de 0, et peuvent ainsi être positives ou négatives. Les valeurs en décibel forment une échelle logarithmique absolue, mais peuvent être également utiles pour la plupart des opcodes. On peut convertir les amplitudes de et en décibel en utilisant les fonctions *ampdb*, *ampdbfs*, *dbamp* et *dbfsamp*. De cette manière, Csound permet au programmeur d'exprimer toutes les amplitudes en dB - les amplitudes plus faibles seront alors représentées par des valeurs de décibel négatives. Cela reflète les pratiques de l'industrie (par exemple sur les indicateurs de niveau des tables de mixage, etc).

Par exemple le même niveau de -6dB (la moitié de l'amplitude) ou de -20dB représentent une amplitude linéaire par rapport à 0dbfs comme ceci :

**Tableau 2.  $dB_{FS}$  en relation avec l'amplitude**

$dB_{FS}$	0dbfs = 32767 (par défaut)	0dbfs = 1	0dbfs = 1000 (inhabituel)
0 dB	32767	1	1000
-6 dB	16384	0.5	500
-20 dB	3276.7	0.1	100

Certains utilisateurs de Csound peuvent ainsi avoir l'intention d'exprimer tous les niveaux en  $dB_{FS}$ , et éviter toute confusion ou toute ambiguïté de niveau qui pourrait autrement se produire lorsque des valeurs explicites d'amplitude sont utilisées. L'échelle en décibel reflète la réponse de l'oreille assez fidèlement, et si vous voulez exprimer un niveau vraiment doux, il peut être plus facile et plus expressif d'écrire "-46dB" que "0.005" ou "163.8".

La raison d'utiliser 0dbfs est très simple : la crête numérique est égale au niveau maximum quelque soit la résolution de l'échantillonnage. Si vous définissez un signal à -110dB, il disparaîtra s'il est restitué dans un fichier 16 bit, mais il restera (audible ou non) s'il est restitué en 24 bit ou mieux. Autrement dit,



il y a un plafond fixe mais un plancher mobile - vous pouvez définir des sons aussi doux que vous le voulez (par exemple des queues d'enveloppe), de manière prévisible, et les préserver ou non (sans changer le code de l'orchestre), selon la résolution de leur restitution (dans un fichier ou sur une e/s audio).



## Une note sur l'amplitude numérique, les décibels et l'étendue dynamique

Une approche commode de l'étendue dynamique pour une certaine précision numérique est de calculer l'intervalle en décibels entre la valeur minimale et la valeur maximale pour un échantillon. En général, 1 bit (doublement du niveau) vaut 6dB, donc 16 bit = 96dB.

Ceci n'est pas entièrement exact car les valeurs des échantillons audio sont représentées sur une échelle bipolaire avec des valeurs positives et négatives, et un bit est utilisé pour le signe. Ainsi, puisque les échantillons en entiers sur 16 bit utilisent 1 bit pour le signe et 15 bit pour la valeur, l'intervalle dynamique est de 90dB.

## Audio en temps-réel

L'information suivante concerne en premier lieu l'utilisation de csound à partir de la ligne de commande. Les frontaux implémentent ces caractéristiques de différentes manières, mais leur connaissance est nécessaire dans certains d'entre eux.

Les options *-i* et *-o* sont utilisées pour spécifier une sortie en temps-réel à la place de l'habituelle sortie différée dans un fichier. On utilise *-o dac* pour la sortie en temps-réel et *-i adc* pour l'entrée en temps-réel. Naturellement, on peut utiliser l'un ou les deux selon les possibilités matérielles. On peut aussi spécifier le matériel à utiliser en ajoutant un numéro ou un nom de périphérique au drapeau (voir *-i* et *-o*).

Il peut aussi être nécessaire d'utiliser l'option *-+rtaudio* pour spécifier le pilote d'interface à utiliser. Csound utilise Portaudio par défaut, qui est multi plates-formes et fiable, mais, pour obtenir de meilleures performances, on peut utiliser ALSA et JACK sur linux, et CoreAudio sur Mac. On peut utiliser ASIO sur Windows si la version de Portaudio a été compilée avec le support ASIO.

On peut voir une liste des périphériques disponibles en donnant un numéro de périphérique trop grand, par exemple *-o dac99*. Si vous utilisez Portaudio, ceci indiquera également si ASIO est disponible.

## Tailles de Période & de Tampon

Les tailles de période et de tampon varient beaucoup d'une machine à l'autre. Plus la taille du tampon est petite et plus la latence est courte, mais cela peut causer des interruptions et des clics dans le flux audio. Les options Csound qui contrôlent les tailles de période et de tampon sont respectivement *-b* et *-B*. La taille de tampon dépend du matériel, et des essais peuvent être nécessaires pour trouver l'équilibre optimal entre une faible latence et un flux audio continu. Les valeurs données à *-b* et *-B* doivent être des puissances de deux, et la valeur de *-B* doit surpasser celle de *-b* d'au moins une puissance de deux.

Actuellement, avec *-B* fixé à 512, la latence de la sortie audio est d'environ 12 millisecondes, suffisamment rapide pour un jeu au clavier raisonnablement réactif. On peut même obtenir des latences plus courtes sur certains systèmes.

## Cadence de Contrôle

De faibles valeurs de ksmpls donneront en général une synthèse de meilleure qualité, mais consommeront plus de ressources système. Il n'y a pas de règle absolue pour fixer ksmpls - différents orchestres nécessiteront différentes cadences de contrôle. Un instrument à guide d'onde nécessitera une valeur de ksmpls de 1 (et pourra ne pas convenir au temps-réel), alors qu'une simple synthèse FM pourra fonction-

ner avec de plus grandes valeurs de ksmpls sans dégradation notable du son. Si cette synthèse FM doit jouer une ligne de basse monodique, on peut utiliser une très faible valeur de ksmpls, cependant des clusters de notes plus complexes nécessiteront une valeur de ksmpls plus grande. Un système linux bien réglé devrait même être capable de produire des synthèses polyphoniques complexes avec des valeurs de ksmpls aussi faibles que 4 ou 8. Si l'on a besoin de capacités audio duplex complètes, *-b* doit être un multiple entier de ksmpls. En gardant cela à l'esprit, on peut poser comme règle empirique de n'utiliser que des puissances de deux pour ksmpls.

Certains réglages diffèrent selon la plate-forme. Voir la suite pour de l'informations sur chaque plate-forme.

## Entrées/Sorties en temps-réel sur Linux

Sous linux, les réglages PortAudio/PortMidi par défaut provoquent une latence plus longue que celle que l'on obtiendrait avec ALSA et/ou JACK. Les plugins PortMusic sont des serveurs audio et MIDI, qui fournissent une interface aux pilotes ALSA, tout comme le fait JACK, mais d'une manière fondamentalement différente. Pour une comparaison plus détaillée prière de se référer à :

<http://jackaudio.org/faq>

## Utilisation d'ALSA

Le plus haut niveau de contrôle et la plus faible latence possible sont atteints en utilisant les plugins ALSA en combinaison avec l'option *--sched*. L'utilisation de *--sched* nécessite que Csound soit lancé par l'utilisateur root, ce qui peut être impossible ou indésirable dans certaines circonstances.

Les plugins ALSA nécessitent le nom ("name") d'une carte ("card") et d'un périphérique ("device"). A moins d'avoir nommé vos cartes dans *~/asoundrc* (ou */etc/asound.conf*), les noms seront en fait des nombres. Pour obtenir une liste des configurations possibles, utilisez les utilitaires en ligne de commande "aplay", "arecord" et "amidi". Ces utilitaires sont inclus dans la plupart des distributions Linux, ou peuvent être téléchargés et construits à partir de ces sources :

<ftp://ftp.alsa-project.org/pub/utils/>

## Sortie Audio

En tapant la commande suivante :

```
aplay -l
```

vous obtiendrez une liste des périphériques de reproduction audio disponibles sur votre système. Cette liste ressemble à ceci :

```
[....]
**** List of PLAYBACK Hardware Devices ****
card 0: A5451 [ALI 5451], device 0: ALI 5451 [ALI 5451]
[....]
```

Si vous avez plus d'une carte sur votre système, ou s'il y a plus d'un périphérique sur votre carte, la liste sera naturellement plus compliquée, cependant, dans tous les cas, l'information pertinente est le numéro/nom de la carte/périphérique. Afin d'utiliser la carte son ci-dessus pour la sortie audio, il faut ajouter l'option suivante à la ligne de commande Csound, dans *~/csoundrc*, ou dans la section *<CsOptions>* d'un CSD :

```
--rtaudio=alsa -o dac
```

## Sortie avec dmix

Si vous désirez utiliser Csound avec dmix et que votre carte son ne supporte pas le mixage matériel des flux audio, il faut régler les tampons logiciel (-b) et matériel (-B) avec un soin particulier. Si vous recevez un message du pilote ALSA de Csound qui ressemble à ceci :

```
ALSA: -B 8192 not allowed on this device; use 7526 instead
```

il y a de bonnes chances que vous puissiez utiliser dmix. Si vous utilisez dmix, les réglages de -b et de -B dans Csound doivent être synchronisés avec la taille de période (period\_size) et la taille de tampon (buffer\_size) de dmix respectivement, en utilisant le rapport du taux d'échantillonnage du projet Csound sur le taux d'échantillonnage sur lequel dmix est réglé. Les formules suivantes déterminent les réglages à utiliser pour Csound en fonction des réglages de dmix :

```
-b = (csound_sr/dmix_sample_rate) * dmix_period_size  
-B = (csound_sr/dmix_sample_rate) * dmix_buffer_size
```

Par exemple, si dmix est fixé à 48000 échantillons par seconde, un period\_size de 1024, et un buffer\_size de 8192, si l'on exécute un projet Csound avec sr=48000, les réglages des tampons seront "-b 1024 -B8192". Si sr=24000, les réglages des tampons seront "-b 512 -B4096".

A cause de cette relation, si le taux d'échantillonnage du projet Csound ne divise pas exactement le taux d'échantillonnage utilisé par dmix, il pourra être difficile, voire impossible, de régler correctement -b et -B à cause des erreurs d'arrondi. En conséquence, si vous utilisez des taux d'échantillonnage différents que ceux que vous fixez pour dmix, nous vous suggérons de configurer dmix avec un period\_size et un buffer\_size divisibles par le rapport entre le taux d'échantillonnage de csound et celui de dmix. Par exemple, pour exécuter un projet avec sr=16000, les réglages suivants de dmix :

```
pcm.amix {  
    type dmix  
    ipc_key 50557  
    slave {  
        pcm "hw:0,0"  
        period_time 0  
        #period_size 1024  
        #buffer_size 8192  
        period_size 1536  
        buffer_size 12288  
    }  
    bindings {  
        0 0  
        1 1  
    }  
}  
  
# route ALSA software through pcm.amix  
pcm.!default {  
    type plug  
    slave.pcm "amix"  
}
```

avec period\_size=1536 et buffer\_size=12288 seront divisibles par 3 (le rapport du taux d'échantillonnage de csound par celui de dmix) pour obtenir "-b 512 -B4096" ((16000/48000) \* 1536 = 512, (16000/48000) \* 12288 = 4096).



### Note

Pour la plupart des cartes son qui sont affectées par ceci, le taux d'échantillonnage par défaut de la carte sera 48000 et ceux de dmix seront 1024 et 8192.

## Entrée Audio

Normalement, la même carte étant utilisée pour les entrées et les sorties, en continuant l'exemple précédent, l'option :

```
-i adc:hw:0,0
```

sera ajouté pour l'entrée audio à partie du périphérique 0 de la carte 0. Pour utiliser la carte par défaut, on emploie l'option suivante, mais attention, ça peut ne pas fonctionner :

```
-i adc
```

Si l'on désire utiliser une autre carte ou un autre périphérique pour l'entrée, la commande suivante fournira une liste de périphériques en entrée :

```
arecord -l
```

Si, par exemple, vous désirez utiliser en sortie une interface audio USB, qui est la deuxième "carte" dans votre système, alors que vous désirez utiliser en entrée votre carte son interne, la première carte de votre installation, positionnez les options suivantes à l'endroit adéquat :

```
--rtaudio=alsa -i adc:hw:0,0 -o dac:hw:1,0
```

Si vous désirez utiliser le second périphérique sur votre interface USB, pour envoyer un flux audio à un canal particulier, vous utiliserez les options suivantes :

```
--rtaudio=alsa -i adc:hw:0,0 -o dac:hw:1,1
```

## Entrée MIDI

Csound ne crée pas automatiquement son propre port de séquenceur ALSA. Il crée un port midi direct ALSA à chaque lancement. Afin de permettre à votre orchestre de recevoir une entrée MIDI vous pouvez utiliser VirMIDI ou MIDITHru, selon vos préférences. La configuration de ces ports MIDI virtuels a été largement couverte ailleurs, voir le Linux MIDI how-to [<http://www.midi-howto.com/>] ou parcourez la documentation de votre distribution ou la documentation ALSA à la recherche d'instructions pour installer et configurer VirMidi ou MIDITHru. Une fois ceci réalisé, la commande :

```
amidi -l
```

retourne une liste des périphériques disponibles. Cette liste ressemble à ceci :

```
[....]
Device  Name
hw:1,0  Virtual Raw MIDI (16 subdevices)
hw:1,1  Virtual Raw MIDI (16 subdevices)
hw:1,2  Virtual Raw MIDI (16 subdevices)
hw:1,3  Virtual Raw MIDI (16 subdevices)
hw:2,0,0 PCR MIDI
hw:2,0,1 PCR 1
```

Dans cet exemple, Csound peut se connecter à n'importe lequel des quatre ports virtuels MIDI directs,

pour y écouter l'entrée MIDI. L'option suivante indique à Csound d'écouter sur le premier de ces ports :

```
--rtmidi=alsa -Mhw:1,0
```

Il faudra ensuite connecter votre matériel ou votre contrôleur logiciel au port qui accueille votre synthétiseur Csound. La manière la plus simple de le faire est d'employer l'utilitaire "aconect". Tapez :

```
aconect -li
```

pour une liste des périphériques d'entrée disponibles, et :

```
aconect -lo
```

pour une liste des périphériques de sortie disponibles (y compris le port auquel Csound a été connecté). Cette liste ressemble à ceci :

```
#aconect -li
client 0: 'System' [type=kernel]
  0 'Timer'
  1 'Announce'
    Connecting To: 15:0
client 20: 'Virtual Raw MIDI 1-0' [type=kernel]
  0 'VirMIDI 1-0'
client 21: 'Virtual Raw MIDI 1-1' [type=kernel]
  0 'VirMIDI 1-1'
client 22: 'Virtual Raw MIDI 1-2' [type=kernel]
  0 'VirMIDI 1-2'
client 23: 'Virtual Raw MIDI 1-3' [type=kernel]
  0 'VirMIDI 1-3'
client 24: 'PCR' [type=kernel]
  0 'PCR MIDI'
  1 'PCR 1'
  2 'PCR 2'
```

```
#aconect -lo
client 20: 'Virtual Raw MIDI 1-0' [type=kernel]
  0 'VirMIDI 1-0'
client 21: 'Virtual Raw MIDI 1-1' [type=kernel]
  0 'VirMIDI 1-1'
client 22: 'Virtual Raw MIDI 1-2' [type=kernel]
  0 'VirMIDI 1-2'
client 23: 'Virtual Raw MIDI 1-3' [type=kernel]
  0 'VirMIDI 1-3'
client 24: 'PCR' [type=kernel]
  0 'PCR MIDI'
  1 'PCR 1'
```

Dans l'exemple suivant, le clavier USB qui est listé ci-dessus comme le client 24 sera connecté au synthétiseur Csound qui est à l'écoute sur le premier port VirMIDI. Le clavier a trois ports de sortie. Le premier (24:0) transmet les messages reçus sur le port d'entrée MIDI, le second (24:1) transmet les messages de touches et de contrôleurs, et le troisième (24:2) transmet les messages système exclusif. La

commande suivante connecte le second port du clavier au synthétiseur Csound :

```
aconnect 24:1 20:0
```

Il faut garder à l'esprit que Csound agit comme un périphérique MIDI direct et non comme un client du séquenceur ALSA. Cela signifie que Csound n'apparaîtra pas dans la liste des périphériques MIDI et ne sera pas disponible pour un usage direct avec *aconnect*, ainsi, il faut se connecter à un périphérique virtuel (comme 'virtual raw MIDI' ou 'MIDI through') pour des connexions persistantes, ou se connecter directement à la destination en utilisant les options de ligne de commande.

## Sortie MIDI

On peut connecter Csound à n'importe quel périphérique qui apparaît dans la liste des ports de sortie du séquenceur ALSA, que l'on obtient par "amidi -l" comme ci-dessus. Afin de connecter un synthétiseur Csound au port MIDI out du clavier listé ci-dessus, on utilise l'option suivante :

```
-Qhw:2,0,0
```

## Temps-partagé

Si vous avez la possibilité d'exécuter Csound en tant qu'utilisateur root, l'option "--sched" permet d'améliorer spectaculairement les performances temps-réel avec ALSA, cependant vous pouvez bloquer le système si vous faites quelque chose de stupide. N'UTILISEZ PAS "--sched" si vous choisissez JACK pour la sortie audio. JACK contrôle le temps-partagé pour les applications audio qui l'utilisent, et il essaie également de fonctionner avec la priorité maximale. Si l'option "--sched" est utilisée, Csound et JACK vont entrer en compétition au lieu de coopérer, ce qui aura pour résultat de piètres performances.

## Utiliser JACK

La manière la plus simple d'activer les entrées-sorties avec le plugin JACK est :

```
--rtaudio=jack -i adc -o dac
```

En outre, il y a quelques options de ligne de commande spécifiques à JACK :

### Options de ligne de commande de JACK

--jack\_client=[nom\_de\_client]

Le nom de client par défaut de Csound est 'csound5'. Si plusieurs instances de Csound se connectent au serveur JACK, il faut utiliser des noms de client différents pour éviter les conflits de noms.

--jack\_inportname=[préfixe du nom de port d'entrée], -  
--jack\_outportname=[préfixe du nom de port de sortie]

Le préfixe du nom des ports d'entrée/sortie JACK de Csound ; la valeur par défaut est 'input' et 'ouput'. Le nom complet d'un port est obtenu en ajoutant le numéro du canal au préfixe. Exemple : avec les réglages par défaut, un orchestre stéréo créera les ports suivants en mode d'opération full duplex :

csound5:input1	(enregistrement gauche)
csound5:input2	(enregistrement droite)
csound5:output1	(reproduction gauche)
csound5:output2	(reproduction droite)

en microsecondes]

Depuis Csound version 5.01, cette option est dépréciée et ignorée.

## Connecter Csound à d'autres clients JACK

Il n'y a par défaut aucune connection (on doit utiliser `jack_connect`, `qjackctl`, ou un utilitaire semblable) ; cependant, on peut connecter le plugin à des ports spécifiés par `'-iadx:portname_prefix'` ou `'-odac:portname_prefix'`, où `portname_prefix` est le nom d'un port sans le numéro de canal, tel que `'alsa_pcm:capture_'` (pour `-i adc`), ou `'alsa_pcm:playback_'` (pour `-o dac`).

## Notes sur les tailles de tampon

Les données audio sont reçues de et envoyées vers le serveur JACK par Csound au moyen d'un tampon circulaire qui est contrôlé par les options `-b` et `-B`. `-B` est la taille totale du tampon, tandis que `-b` est la taille d'une période. Ces valeurs sont arrondies de façon à ce que la taille totale soit un multiple entier de la taille de la période et supérieure à cette dernière. La différence de taille entre le tampon de Csound et la période doit être supérieure ou égale à la taille de la période de JACK.

Si l'on utilise en même temps `-iadc` et `-odac`, l'option `-b` doit être fixée à une valeur multiple de `ksmps`.

Exemple de réglage de tampon pour obtenir une faible latence sur un système rapide :

```
jackd -d alsa -P -r 48000 -p 64 -n 4 -zt &
csound --rtaudio=jack -b 64 -B 256 [...]
```

avec temps-partagé pour le temps-réel (en tant que root) :

```
jackd -R -P 90 -d alsa -P -r 48000 -p 64 -n 2 -zt &
csound --sched=80,90,10 -d --rtaudio=jack -b 64 -B 192 [...]
```

Pour améliorer les performances, utiliser des valeurs de `ksmps` comme 32 ou 64.

Le taux d'échantillonnage de l'orchestre doit être le même que celui du serveur JACK.

## Utilisation de Pulseaudio

Le support de Pulseaudio [<http://www.pulseaudio.org/>] a été ajouté dans Csound 5.09. Vous pouvez spécifier les réglages suivants :

1. Noms de sortie : il est possible d'utiliser un nombre à la place du nom complet, ainsi `-odac:1` sélectionne votre second périphérique (le compte commence à 0).
2. Nom du serveur : il est possible de se connecter à un serveur spécifique en utilisant `-+server=<server_string>` où `server_string` est le nom d'un serveur ou une chaîne plus complexe de sélection de serveur (voir [pulseaudio.org](http://www.pulseaudio.org/) [<http://www.pulseaudio.org/>] sur les chaînes de serveur). Ceci est transparent sur un réseau et permet les connexions à des machines distantes.
3. Noms de flot : il est possible d'étiqueter les flots générés par `csound`, en utilisant `-+output_stream=<stream-name>` et `-+input_stream=<stream-name>`

Voici un exemple de ligne de commande :

```
csound -odac:1 examples/trapped.csd --rtaudio=pulse --server=unix:/tmp/pulse-victor/native --output_stream=trapped
```

## Windows

## Audio en temps-réel

Les utilisateurs de Windows peuvent utiliser soit le module temps-réel par défaut *PortAudio*, soit le module temps-réel *winmm*. Le module winmm est un module natif de Windows qui fournit une grande stabilité, mais une latence qui sera en général trop grande pour une interaction en temps réel. Pour activer un module temps-réel on peut utiliser l'option `-+rtaudio` avec la valeur *portaudio* ou *winmm*. La valeur par défaut est *portaudio*, qui est active sans avoir à être spécifiée.

On doit aussi spécifier le périphérique son que l'on veut utiliser, et indiquer que l'on veut générer une sortie audio en temps-réel plutôt qu'un fichier son vers une sortie disque. Pour cela, on doit utiliser l'option `-odac` ou `-o dac`, qui indique comme sortie de csound les convertisseurs Numérique-Analogique plutôt qu'un fichier. En ajoutant un numéro après l'option (par exemple `-odac2`), on peut choisir le numéro du périphérique désiré. Pour trouver les périphériques disponibles dans le système, on peut utiliser un numéro trop grand (par exemple `-odac99`), et csound rapportera une erreur ainsi que la liste des périphériques disponibles.

Lorsque l'on choisit le numéro de périphérique sous Portaudio, on choisit également l'interface du pilote, car Portaudio supporte WinMME, DirectX et ASIO. Si vous avez une interface compatible ASIO ou un émulateur de pilote ASIO comme ASIO4ALL [<http://www.asio4all.com>], le périphérique affichera plusieurs durées, une pour chaque interface de pilote. Comme ASIO fournit la meilleure latence pour un système, il devrait être choisi pour une sortie audio en temps-réel s'il est disponible.

On active l'entrée audio en temps-réel par `-iadc`, ce qui règle csound sur l'écoute de l'entrée audio temps-réel. On peut également choisir le périphérique par son numéro, et tester les périphériques disponibles avec un numéro trop grand. Notez que pour les entrées on utilise 'adc' au lieu de 'dac'. Assurez-vous que la bonne entrée soit sélectionnée dans le panneau de contrôle de votre carte son.

## MIDI en temps-réel

Pour activer le MIDI en temps-réel dans Windows on peut utiliser l'option `-M` pour l'entrée MIDI et l'option `-Q` pour la sortie MIDI. On peut spécifier le numéro du périphérique après le drapeau (par exemple `-M2`), et aussi trouver les périphériques disponibles en donnant un numéro trop grand.

Csound utilise par défaut le module MIDI PortMidi, mais il y a aussi un module natif winmme, que l'on peut activer avec l'option :

```
--rtmidi=winmme
```

Un ensemble d'options typique pour activer l'Audio et les E/S MIDI en temps-réel ressemblera à ceci:

```
--rtmidi=winmme -M1 -Q1 --rtaudio=portaudio -odac3 -iadc3
```

## Mac

Prochainement...

## Optimisation de la Latence Audio en E/S

Pour atteindre la latence la plus basse possible sans interruptions audio, il faut régler une combinaison de variables. Les valeurs retenues dépendront de la plate-forme et du système, et aussi de la complexité



des calculs audio mis en œuvre. Il faut ajuster *ksmps* dans l'orchestre, ainsi que la taille du tampon logiciel (*-b*) et celle du tampon matériel (*-B*).

Habituellement la solution la plus simple est la suivante :

1. Fixer *ksmps* à une valeur de compromis entre qualité et performance, sans ajuster *-B* du tout.
2. Fixer *-b* à une puissance de deux négative.

Pour obtenir les valeurs optimales, commencer avec une valeur qui vous semble trop petite, c'est-à-dire -1, et continuer ensuite en "augmentant", -2, -4, etc., jusqu'à ne plus avoir de défauts dans le son. La valeur réelle de *-b* sera la valeur absolue de  $-b * ksmps$ .

3. Réduire le tampon matériel (*-B*). Partir de la valeur par défaut (1024 sur Linux, 4096 sur Max OS X, 16384 sur Windows), et la réduire de moitié à chaque fois, jusqu'à entendre à nouveau des défauts. La remonter alors jusqu'à ce que l'exécution soit continue.

Cette procédure s'applique aux cartes 16 bit. Si vous avez une carte son 24 bit, alors *-B* doit valoir 3/2, ou 3 fois *-b*, plutôt que 2 ou 4 fois. Csound travaille avec des nombres en virgule flottante en 32 bit ou 64 bit alors que la plupart des cartes son utilisent des entiers en 16 ou 24 bit. *-b* est le tampon interne, c'est pourquoi il traite de la partie 32 ou 64 bit, tandis que *-B* est le tampon matériel, et il traite ainsi de la partie 16 ou 24 bit. Le réglage par défaut de csound pour les réels est  $-B = 4 * -b$ . C'est une valeur sûre pour une carte 16 bit. On peut s'en sortir avec  $-B = 2 * -b$ , mais c'est le minimum absolu. Par exemple, si votre réglage est *-b1024 -B2048*, csound vous dira ceci :

```
audio buffered in 1024 sample-frame blocks
writing 4096-byte blocks to dac
```

4096 octets font 32768 bits.  $32768/32 = 1024$ , notre taille de bloc de trames d'échantillons,  $1024 * 32/16 = 2048$ , notre taille de tampon. Si nous réduisons la valeur de *-B*, il faudra réduire la valeur de *-b* d'un montant proportionnel afin de continuer à écrire des entiers en 16 bit sur le CNA. La taille minimale de *-b* est  $(-B * bitrate)/32$ . Cela veut dire que le rapport minimum de *-b* à *-B* doit être :

- 1/2 en 16 bit
- 2/3 en 24 bit
- 1/1 en 32 bit

Bien qu'il n'y ait théoriquement pas de rapport maximum, il n'y a aucun sens à avoir un rapport très élevé ici, car le tampon logiciel doit remplir le tampon matériel avant de retourner. Si le rapport est élevé, cela prendra plus de temps, annulant le bénéfice de mettre une petite valeur pour *-b*.

Il faudra varier la valeur de *-b* en fonction de la complexité de l'instrument sur lequel vous travaillez, mais comme elle est intimement liée à celle de *ksmps*, il vaut mieux la synchroniser avec *ksmps* et partir de là. Une manière de faire est de décider quelle sera la longueur optimale de la chute de vos enveloppes (pour l'effet désiré), de fixer toutes les enveloppes au maximum, de donner vous-même une valeur généreuse à *-b*, et de jouer. S'il y a des interruptions, doubler *ksmps*, et répéter le processus jusqu'à obtenir la fluidité, descendre ensuite la valeur de *-b* aussi bas que possible.

La valeur de *-B* est d'abord déterminée par le système d'exploitation et la carte son. Essayez de trouver (par la méthode ci-dessus) jusqu'où vous pouvez descendre, et utilisez cette valeur (ou une valeur supérieure par sécurité). Si vous rencontrez des problèmes ce sera probablement à cause d'une valeur de *ksmps* inappropriée, d'une valeur de *-b* trop faible, ou de nombres hors-norme (voir *denorm*).

---

# Configuration

Après avoir installé une distribution binaire ou bien avoir construit Csound à partir des sources, il faut configurer Csound afin de l'adapter à votre système. Les installateurs réalisent habituellement ces étapes automatiquement pour vous.

Sur toutes les plates-formes il faut s'assurer que le ou les répertoires contenant les bibliothèques des plugins de Csound sont indiqués dans une variable d'environnement `OPCODEDIR` ou `OPCODEDIR64` en fonction de la précision utilisée par les binaires compilés.

Les opérateurs Python nécessitent actuellement au moins Python 2.4 que l'on peut télécharger à [www.python.org](http://www.python.org) [<http://www.python.org>] s'il n'est pas déjà installé sur votre système. On peut tester s'il est installé en tapant 'python' depuis une invite de commande ou une fenêtre DOS.

## Windows

Sur Windows, assurez-vous que le ou les répertoires (normalement le répertoire `C:\Program Files\Csound`) contenant le répertoire des exécutables de Csound est dans votre variable `PATH`, ou bien copiez tous les fichiers exécutables dans le répertoire `system32` de Windows. En fonction de votre méthode d'installation, il peut être aussi nécessaire de fixer les variables d'environnement `OPCODEDIR` et `OPCODEDIR64`. En supposant que Csound est installé dans le répertoire par défaut `C:\Program Files\Csound` vous pouvez utiliser (sinon fixez les chemins en conséquence) :

```
set OPCODEDIR=C:\Program Files\Csound\plugins
set OPCODEDIR64=C:\Program Files\Csound\plugins64
set PATH=%PATH%;C:\Program Files\Csound\bin
```



### python24.dll ou python25.dll manquante

S'il apparaît une fenêtre pop-up au sujet de la bibliothèque Python manquante (`python24.dll` ou `python25.dll`) et que vous n'avez pas besoin des opérateurs python, effacez simplement `C:\Program Files\Csound\plugins\py.dll` et `C:\Program Files\Csound\plugins64\py.dll`, et la fenêtre pop-up au sujet de la bibliothèque Python manquante ne devrait plus réapparaître

## Unix et Linux

Sur Unix et Linux, installez le programme Csound dans l'un des répertoires `bin` du système, normalement `/usr/local/bin`, et les bibliothèques partagées de Csound et des plugins dans des endroits comme `/usr/local/lib/csound/plugins` ou `/usr/local/lib/csound/plugins64` et assurez-vous que les variables d'environnement `OPCODEDIR` et `OPCODEDIR64` sont remplies correctement.

## CsoundAC

CsoundAC nécessite quelques configurations supplémentaires. Sur toutes les plates-formes, CsoundAC nécessite que vous ayez installé Python sur votre ordinateur. Le répertoire contenant la bibliothèque partagée `_CsoundAC` et le fichier `CsoundAC.py` doit être dans votre variable d'environnement `PYTHONPATH`, afin que le runtime Python sache comment charger ces fichiers.

---

# Syntaxe de l'Orchestre

L'orchestre Csound (.orc) ou la section `<CsInstruments>` d'un fichier csd, contient :

- Une *section d'en-tête*, qui spécifie les options globales pour l'exécution des instruments.
- Une liste d'*opcodes définis par l'utilisateur (UDO)* et de *blocs d'instrument* contenant les définitions des UDO et des instruments.

L'en-tête de l'orchestre, les blocs d'instrument, et les UDOs contiennent des *instructions d'Orchestre*. Dans Csound une *instruction d'orchestre* a le format :

```
étiquette:   résultat opcode argument1, argument2, ... ;commentaires
```

L'étiquette est facultative et indentifie l'instruction de base qui suit comme cible potentielle d'une opération goto (voir *Contrôle du Déroulement du Programme*). Une étiquette n'a aucun effet sur l'instruction en soi.

Selon leur fonction, certains opcodes ne produisent pas de sortie et n'ont donc pas de valeur de retour. D'autres ne prennent pas d'argument et produisent seulement un résultat.

Chaque instruction d'orchestre doit tenir sur une seule ligne, cependant les longues lignes peuvent être continuées sur la ligne suivante grâce au caractère `\`. Ce caractère indique que la ligne suivante fait partie de la ligne courante, de façon à pouvoir couper une ligne pour en faciliter la lecture, comme ceci :

```
a2  oscbnk  kcps, 1.0, kfmdl, 0.0, 40, 203, 0.1, 0.2, kamfr, kamfr2, 148, \
      0, 0, 0, 0, 0, 0, -1, \
      kfnum, 3, 4
```

Les commentaires sont facultatifs et ils ont pour but de permettre à l'utilisateur de commenter le code de son orchestre. Les commentaires commencent par un point-virgule (;) et s'étendent jusqu'à la fin de la ligne. Les commentaires peuvent optionnellement être écrits en style C, s'étendant sur plusieurs lignes comme ceci :

```
/* Tout ce qui se trouve ici -----
   est un commentaire qui peut couvrir
   plusieurs lignes ----- */
```

Le reste (résultat, opcode, et arguments) forme l'instruction de base. C'est également facultatif, ce qui veut dire qu'une ligne peut n'avoir qu'une étiquette ou un commentaire ou bien être complètement blanche. Si elle est présente, l'instruction de base doit être entièrement contenue dans une ligne, et elle est terminée par un retour chariot et un linefeed.

L'opcode détermine l'opération à effectuer ; habituellement, il prend un certain nombre de valeurs en entrée (ou arguments, au maximum environ 800) ; et il a normalement un champ résultat variable dans lequel il envoie les valeurs de sortie à un certain taux de cadencement fixe. Il y a quatre taux de cadencement possibles :

1. une seule fois, au moment de l'initialisation de l'orchestre (en fait une affectation permanente)

2. une fois au début de chaque note (à la date (init) de l'initialisation : taux-i)
3. à chaque passage dans la boucle de contrôle de l'exécution (taux de contrôle, ou taux-k)
4. à chaque échantillon sonore de chaque boucle de contrôle (taux d'exécution audio, ou taux-a)

## Instructions de l'En-tête de l'Orchestre

L'*En-tête de l'Orchestre* contient l'information globale qui s'applique à tous les instruments et qui définit les aspects de la sortie de Csound. On y fait parfois référence comme *instr 0*, parce qu'il se comporte comme un instrument, mais sans traitement de taux-k ou de taux-a (seuls les opcodes et les instructions qui fonctionnent au taux-i y sont autorisés).

Une *instruction d'en-tête d'orchestre* n'opère qu'une fois, à l'initialisation de l'orchestre. La plupart du temps il s'agit de l'affectation d'une valeur à un *symbole global réservé*, par exemple `sr = 20000`. Toutes les instructions d'en-tête d'orchestre appartiennent au pseudo instrument 0, dont un passage *init* est effectué avant tout autre instrument au temps 0 de la partition. Toute *instruction ordinaire* peut servir d'instruction d'en-tête d'orchestre, par exemple `gifreq = cpspch(8.09)` à condition d'être seulement une opération du moment d'initialisation. Les instructions placées normalement dans un en-tête d'orchestre sont :

- *0dbfs*
- *ctrlinit*
- *ftgen*
- *kr*
- *ksmps*
- *massign*
- *nchnls*
- *pgmassign*
- *pset*
- *seed*
- *sr*
- *strset*

Par exemple, un en-tête de Csound peut ressembler à ceci :

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

massign 1, 10
```

## Instructions de Bloc d'Instrument et d'Opcode

Un bloc d'instrument comprend des *instructions ordinaires* qui fixent des valeurs, contrôlent le déroulement logique, ou appellent les différents sous-programmes de traitement du signal qui mènent à la sortie audio. Les instructions qui définissent un bloc d'instrument sont :

- *instr*
- *endin*

Un bloc d'instrument ressemble à ceci :

```
instr 1 ; Un simple oscillateur sinusoïdal
aout oscils 10000, 440, 0
out aout
endin
```

Les instructions qui définissent un bloc d'opcode défini par l'utilisateur (UDO) sont :

- *opcode*
- *endop*

voir la section *UDO* pour plus d'information.

## Instructions Ordinaires

On utilise une *instruction ordinaire* soit lors de l'initialisation soit pendant l'exécution soit durant les deux. Les opérations qui produisent un résultat fonctionnent formellement au taux de ce résultat (c'est-à-dire, pendant l'initialisation pour les résultats de taux-i ; pendant l'exécution pour les résultats de taux-k et de taux-a), avec pour seule exception l'opcode *init*. Cependant, la plupart des générateurs et des modificateurs produisent des signaux que ne dépendent pas seulement de la valeur instantanée de leurs arguments mais aussi d'un état interne conservé. Ainsi, ces unités de la période d'exécution ont un composant implicite de la période d'initialisation pour créer cet état. Le type temporel d'une opération qui ne produit pas de résultat est apparent dans l'opcode.

Les arguments sont des valeurs qui sont envoyées à une opération. La plupart des arguments accepteront des expressions arithmétiques composées de constantes, de variables, de symboles réservés, de convertisseurs de valeur, d'opérations arithmétiques, et de valeurs conditionnelles.

## Types, Constantes et Variables

Les *constantes* sont des nombres en virgule flottante tels que 1, 3.14159 ou -73.45. Elles sont constamment disponibles et leur valeur ne change pas.

Les *variables* sont des cellules nommées contenant des nombres. Elles sont constamment disponibles et peuvent être mises à jour à l'un des quatre taux de mise à jour (initialisation seulement, taux-i, taux-k, taux-a). Les variables de taux-i et de taux-k sont scalaires (c'est-à-dire qu'elles ne peuvent prendre qu'une valeur à la fois) et sont utilisées principalement pour stocker et rappeler des données de contrôle, données qui changent au rythme des notes (pour les variables de taux-i) ou au taux de contrôle (pour les variables de taux-k). Les i- et les k-variables sont ainsi utiles pour stocker les valeurs des paramètres de note, hauteurs, durées, fréquences variant lentement, vibratos, etc. D'un autre côté, les variables de taux-a sont des tableaux ou vecteurs d'information. Bien que rafraichies pendant le même passage de contrôle de la période d'exécution que les variables de taux-k, ces cellules de tableau représentent une résolution temporelle plus fine en divisant la période de contrôle en durées d'échantillons (voir *ksmps*). Les variables de taux-a sont utilisées pour stocker et rappeler des données qui changent au taux

d'échantillonnage audio (par exemple les signaux de sortie des oscillateurs, des filtres, etc.).

Certains types de variables peuvent être considérés comme des signaux. Par exemple les variables de *taux-a* et de *taux-k* sont des signaux qui ont une fréquence de mise à jour constante (voir *kr* et *sr*). Cette abstraction est en général assez utile, mais il faut être conscient que les signaux de *taux-a* sont en fait des vecteurs qui sont traités au *taux-k*, c'est-à-dire que Csound travaille en interne au *taux-k* mais qu'il traite *ksmps* échantillons numériques pour chaque variable de *taux-a* à chaque cycle de contrôle.

Il y a d'autres types de signaux qui nécessitent des *taux* qui ne concordent pas avec *kr* ou *sr*. Les signaux de *taux-f* et de *taux-w* sont utilisés pour le traitement spectral et leur *taux* est déterminé par la taille de fenêtre et le facteur de recouvrement.

On distingue également les variables locales des variables globales. Les variables *locales* sont privées dans un instrument, et un autre instrument ne peut y accéder ni en lecture ni en écriture. Leurs valeurs sont conservées, et leur information est reportée de passage en passage (par exemple de la période d'initialisation à la période d'exécution) à l'intérieur d'un instrument. Les noms de variable locale commencent par la lettre *p*, *i*, *k*, ou *a*. Le même nom de variable locale peut apparaître dans plusieurs blocs d'instrument différents sans conflit.

Les variables *globales* sont des cellules qui sont accessibles par tous les instruments. Leurs noms sont formés soit comme les noms locaux précédés de la lettre *g*, soit de symboles réservés spéciaux. Les variables globales sont utilisées pour diffuser des valeurs générales, pour la communication entre instruments (sémaphores), ou pour envoyer un son d'un instrument à l'autre (par exemple un mixage avant une réverbération).

Etant données ces distinctions, il y a huit formes de variables locales et globales :

**Tableau 3. Types de Variables**

Type	Moment de Renouvellement	Local	Global
symboles réservés	permanent	--	rsymbole
p-champs de partition	temps-i	p nombre	--
variables d'initialisation	temps-i	i nom	gi nom
signaux de contrôle	temps-p, <i>taux-k</i>	k nom	gk nom
signaux audio	temps-p, <i>taux-k</i> (tous les échantillons audio dans une passe-k)	a nom	ga nom
types de données spectrales	<i>taux-k</i>	w nom	--
flots de données spectrales	<i>taux-k</i>	f nom	gf nom
variables chaînes	temps-i et optionnellement temps-k	S nom	gS nom

où *rsymbole* est un symbole réservé spécial (par exemple *sr*, *kr*), *nombre* est un entier positif faisant référence à un p-champ de partition ou à un numéro de séquence, et *nom* est une chaîne composée de lettres, du caractère de soulignement, et/ou de chiffres, avec une signification locale ou globale. Comme on peut le voir, les paramètres de partition sont des variables de *taux-i* dont les valeurs sont copiées à partir de l'instruction de partition appelante juste avant la passe d'initialisation d'un instrument, tandis que les contrôleurs MIDI sont des variables que l'on peut mettre à jour de manière asynchrone depuis un fichier MIDI ou un périphérique MIDI.

## Initialisation de Variable

Les opcodes qui permettent l'initialisation de variable sont :

- *assign*
- *divz*
- *init*
- *tival*

## Macros de Constantes Mathématiques Prédéfinies

Csound définit plusieurs constantes mathématiques importantes par des *Macros*. On peut consulter la liste complète *ici*.

## Expressions

On peut composer des expressions de n'importe quelle profondeur. Chaque partie d'une expression est évaluée à son propre taux. Par exemple, si tous les termes d'une sous-expression changent au taux de contrôle ou plus lentement, cette sous-expression ne sera évaluée qu'au taux de contrôle ; le résultat peut alors être utilisé dans une évaluation au taux audio. Par exemple, dans

```
k1 + abs(int(p5) + frac(p5) * 100/12 + sqrt(k1))
```

100/12 sera évalué à l'initialisation de l'orchestre, les expressions en p5 seront évaluées à l'initialisation de la note, et le reste de l'expression à chaque période-k. Le tout pourrait apparaître en position d'argument dans un générateur unitaire, ou bien faire partie d'une instruction d'affectation.

## Répertoires et Fichiers

Plusieurs générateurs et la commande Csound elle-même spécifient des noms de fichier pour l'écriture ou la lecture. Ceux-ci peuvent parfois être des chemins complets, dont le répertoire cible est complètement spécifié. Lorsque le chemin n'est pas complet, les noms de fichiers sont recherchés dans plusieurs répertoires dans un ordre dépendant de leur type et de la valeur de certaines variables d'environnement. Ces dernières sont facultatives, mais elles peuvent servir à partitionner et à organiser les répertoires de façon à partager les fichiers plutôt que de les dupliquer dans plusieurs répertoires de l'utilisateur. Les variables d'environnement peuvent définir des répertoires pour les fichiers son (SFDIR), les sons échantillonnés (SSDIR), les analyses de son (SADIR), et les fichiers à inclure pour l'orchestre et la partition (INCDIR).

A partir de la version 5.00 de Csound, ces variables d'environnement peuvent spécifier plusieurs répertoires dans une liste dont le séparateur est le point-virgule (;). Si un fichier est trouvé à plusieurs endroits, c'est le premier de ceux-ci qui a la priorité.

L'ordre de recherche est :

1. Les fichiers son en écriture sont placés dans SFDIR (s'il existe), sinon dans le répertoire courant.
2. Les fichiers son en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne

sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans SSDIR puis dans SFDIR.

3. Les fichiers de contrôle d'analyse en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans SADIR.
4. Les fichiers MIDI en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans MFDIR, SSDIR et SFDIR.
5. Les fichiers de code à inclure dans les fichiers d'orchestre et de partition (avec *#include*) sont recherchés d'abord dans le répertoire courant, ensuite dans le même répertoire que le fichier d'orchestre ou de partition (respectivement), enfin dans INCDIR.

## Nomenclature

Tout au long de ce document, les opcodes sont indiqués en **caractères gras** et les mnémoniques de leurs arguments et de leur résultat, lorsqu'ils sont mentionnés dans le texte, sont écrits en *italique*. Les noms d'arguments sont généralement des mnémoniques (*amp*, *phs*), et le résultat est souvent dénoté par la lettre *r*. Tous commencent par une qualification de type *i*, *k*, *a*, ou *x* (par exemple *kamp*, *iphs*, *ar*). Le préfixe *i* dénote des valeurs scalaires au temps de l'initialisation de note ; les préfixes *k* ou *a* dénotent des valeurs de contrôle (scalaires) et audio (vectorielles), modifiées et référencées en continu tout au long de l'exécution (c'est-à-dire à chaque période de contrôle tant que l'instrument est actif). Les arguments sont utilisés aux temps indiqués par leur préfixe ; les résultats sont créés aux temps de leur préfixe, et restent disponibles ensuite pour être utilisés comme entrées ailleurs. A part quelques exceptions, les taux des arguments ne peuvent pas dépasser le taux du résultat. La validité des entrées est définie comme suit :

- arguments avec préfixe *i* doivent être valides à l'initialisation ;
- arguments avec préfixe *k* peuvent être des valeurs de contrôle ou d'initialisation (qui restent valides) ;
- arguments avec préfixe *a* doivent être des entrées vectorielles ;
- arguments avec préfixe *x* peuvent être soit des vecteurs soit des scalaires (le compilateur distinguera).

Tous les arguments, sauf précision contraire, peuvent être des expressions dont les résultats sont conformes à la liste ci-dessus. La plupart des opcodes (tels que **linen** et **oscil**) peuvent être utilisés dans plusieurs modes, le choix étant déterminé par le préfixe ou le symbole du résultat.

Tout au long de ce manuel, le terme "opcode" est utilisé pour indiquer une commande qui produit habituellement une sortie au taux-*a*, -*k* ou -*i*, et qui forme toujours la base d'une instruction complète d'un orchestre Csound. Des éléments comme "+" ou "*sin(x)*" ou, "( *a* >= *b* ? *c* : *d* )" sont appelés "opérateurs."

## Macros

Les macros de l'orchestre fonctionnent comme les macros du préprocesseur C, et remplacent le contenu de la macro dans l'orchestre avant sa compilation. Les opcodes qui servent à créer, appeler, ou annuler les macros de l'orchestre sont :

- *#define*
- *\$NAME*



- `#ifdef`
- `#ifndef`
- `#end`
- `#else`
- `#include`
- `#undef`

On peut aussi définir des macros de l'orchestre au moyen de l'option de la ligne de commande `--omacro:`.

On peut trouver plus d'information et des exemples sur l'utilisation des macros de l'orchestre à `#define`.

Ces opcodes font référence aux macros de l'orchestre ; pour les macros de la partition, voir *Macros de Partition*.

## Instruments Nommés

La syntaxe de l'orchestre a été modifiée récemment pour permettre de définir des instruments avec des noms en chaîne de caractères. On peut appeler les instruments ainsi nommés depuis la partition et ils sont supportés par un certain nombre d'opcodes.

## Syntaxe

Un instrument nommé est déclaré comme suit :

```
instr Nom[ , Nom2[ , Nom3[ , ... ] ] ]
[ ... ]
endin
```

Un instrument seul peut avoir autant de noms que l'on veut, et chacun de ces noms peut être utilisé pour appeler l'instrument. De plus, il est possible d'utiliser des nombres comme des noms, dénotant un instrument numéroté de façon standard, ce qui fait que la déclaration suivante est également valide :

```
instr 100, Nom1, 99, Nom2, 1, 2, 3
```

Un nom d'instrument est constitué de lettres, de chiffres, et du caractère de soulignement (`_`), sans limite de taille, cependant, le premier caractère ne doit pas être un chiffre. Optionnellement, le nom de l'instrument peut-être préfixé par un caractère `+` (voir ci-dessous), par exemple :

```
instr +Reverb
```

Pour tous les noms d'instrument, un numéro est affecté automatiquement (note : si le niveau des messages (-m) n'est pas nul, ces numéros sont imprimés sur la console pendant la compilation de l'orchestre), en suivant ces règles :

- le nombre est choisi parmi les numéros d'instrument non affectés en ordre ascendant, en commençant par 1

- les numéros sont affectés dans l'ordre de définition des noms d'instrument, si bien que les derniers instruments nommés auront toujours un numéro plus élevé (sauf si le modificateur '+' est utilisé)
- si le nom de l'instrument est préfixé par un '+', le numéro affecté sera plus grand que tous ceux des autres instruments sans le '+' (numérotés et nommés). S'il y a plusieurs instruments '+', la numérotation de ceux-ci suivra l'ordre de leur définition, selon la règle ci-dessus.

L'utilisation de '+' est surtout utile pour la sortie globale ou les instruments d'effets, qui doivent être exécutés après les autres instruments.

Exemple de numérotation d'instruments :

```
instr 1, 2
endin

instr Instr1
endin

instr +Effet1, Instr2
endin

instr 100, Instr3, +Effet2, Instr4, 5
endin
```

Dans cet exemple, les numéros d'instrument sont affectés comme suit :

```
Instr1: 3
Effet1: 101
Instr2: 4
Instr3: 6
Effet2: 102
Instr4: 7
```

## Utilisation des Instruments Nommés

On peut appeler les instruments nommés en utilisant le nom entre guillemets à la place du numéro d'instrument (note : le caractère '+' doit être omis). Actuellement (depuis Csound 4.22.4), les instruments nommés sont supportés par :

- les événements de partition 'i' et 'q'



### Notes

1. dans les fichiers de partition, il faut éviter les guillemets non appariés, et les espaces et autres caractères illégaux dans les chaînes, sinon (au moins dans la version actuelle) un comportement imprévisible peut apparaître (ce problème n'existe pas pour les événements en ligne -L). Cependant, il y a un test pour détecter les instruments non définis, et dans ce cas, l'évènement est simplement ignoré avec un avertissement.
2. Les utilitaires autonomes (scsort et extract) ne supportent pas les instruments nommés. Il est toujours possible de trier de telles partitions en utilisant l'option -t0 de l'exécutable Csound.

- les événement temps-réel en ligne (-L)
- les opcodes event, schedkwhen, subinstr, et subinstrinit

- les opcodes `massign`, `pgmassign`, `prealloc`, et `mute`

De plus, il y a un nouvel opcode (`nstrnum`) qui retourne le numéro d'un instrument nommé :

```
insno nstrnum "nom"
```

Dans l'exemple ci-dessus, `nstrnum` "Effet1" retournerait 101. S'il n'existe aucun instrument avec le nom spécifié, une erreur d'initialisation est levée et -1 est retourné.

## Exemple

```
; ---- orchestre ----

sr      = 44100
ksmps   = 10
nchnls  = 1

prealloc "SineWave", 20
prealloc "MIDISineWave", 20

massign 1, "MIDISineWave"

gaOutSend init 0

instr +OutputInstr

out gaOutSend
clear gaOutSend

endin

instr SineWave

a1 oscils p4, p5, 0
vincr gaOutSend, a1

endin

instr MIDISineWave

iamp veloc
inote notnum
icps = cpsoct(inote / 12 + 3)
a1 oscils iamp * 100, icps, 0
vincr gaOutSend, a1

endin

; ---- partition ----

i "SineWave" 0 2 12000 440
i "OutputInstr" 0 3
e
```

## Auteur

Istvan Varga

2002

## Opcodes Définis par l'Utilisateur (UDO)

Csound permet la définition d'opcodes dans l'en-tête de l'orchestre au moyen des opcodes *opcode* et *endop*. L'opcode défini peut fonctionner avec un nombre d'échantillons par période de contrôle (*ksmps*)

différent en utilisant *setksmps*.

Pour connecter les entrées et les sorties de l'UDO, on utilise *xin* et *xout*.

Un UDO ressemble à ceci :

```

opcode Lowpass, a, akk
    setksmps 1                ; nécessite sr=kr
    ain, kal, ka2 xin          ; lire les paramètres d'entrée
    aout init 0                ; initialiser la sortie
    aout = ain*kal + aout*ka2  ; filtre simple comme tone
    xout aout                  ; écrire la sortie
endop

```

Cet UDO appelé *Lowpass* reçoit trois entrées (la première au taux-a, et les deux autres au taux-k), et délivre une sortie au taux-a. Noter l'utilisation de *xin* pour recevoir les entrées et de *xout* pour délivrer les sorties. Noter aussi l'utilisation de *setksmps*, qui est nécessaire pour que le filtre fonctionne correctement.

Pour utiliser cet UDO depuis un instrument, on écrirait quelque chose comme :

```
afiltre Lowpass asource, kvaieur1, kvaieur2
```

voir l'entrée *opcode* pour des informations détaillées sur la définition d'UDO.

Vous pouvez trouver plusieurs UDO déjà rédigés (ou apporter votre propre contribution) à *User Defined Opcode Database* [<http://www.csounds.com/udo/>] sur *Csounds.com* [<http://www.csounds.com/>].

---

# La Partition Numérique Standard

La section de la partition contient des évènements qui démarrent des instances d'instruments de l'orchestre. La partition propose diverses instructions qui permettent l'élaboration de partitions complexes avec le langage de csound.

Actuellement, la longueur maximale de la partition est de  $2^{31}-1$  périodes de contrôle. Par exemple, avec  $kr=1500$ , on peut exécuter une partition pendant une période maximale de 16,5 jours avant l'apparition de problèmes provoqués par un dépassement des variables entières signées sur 32 bit.

Il faut noter également que lorsque l'on utilise des nombres flottants en simple précision (c-à-d les installeurs 'f' plutôt que les 'd'), la précision temporelle se détériore après une longue durée d'exécution.

## Prétraitement des Partitions Standard

Une *Partition* (un ensemble d'instructions de partition) se divise en sections ordonnées dans le temps par l'*instruction s*. Avant sa lecture par l'orchestre, une partition est prétraitée section par section. Chaque section est normalement traitée par trois routines : *Carry* (report de valeur), *Tempo*, et *Sort* (tri).

### Carry

Dans un groupe d'*instructions i* consécutives dont les nombres entiers p1 sont indentiques, tout p-champ non rempli prendra la même valeur que celle du p-champ correspondant dans l'instruction précédente. Un p-champ vide peut-être marqué par un point (.) entouré d'espaces. Il n'y a pas besoin de point après le dernier p-champ non vide. La sortie du pré-traitement Carry montre explicitement les valeurs reportées. La Fonction Carry n'est pas affectée par les commentaires rencontrés ou les lignes blanches ; elle s'arrête seulement lorsqu'elle rencontre une instruction autre que l'*instruction i* ou une *instruction i* avec un nombre entier p1 différent.

Il y a trois fonctions supplémentaires, pour p2 seulement : +, ^+x, et ^-x. Le symbole + en p2 recevra la valeur de p2 + p3 de l'instruction i précédente. Cela permet de déterminer automatiquement l'instant du début d'une note à partir de la somme des durées précédentes. Le symbole + peut lui-même être reporté. Il n'est autorisé que dans p2. Par exemple : les instructions

```
i1  0      .5    100
i  .  +
i
```

se transformeront en

```
i1  0      .5    100
i1  .5     .5    100
i1  1      .5    100
```

Les symboles ^+x et ^-x déterminent la valeur de p2 en additionnant ou en soustrayant respectivement la valeur x du p2 précédent. Ils ne peuvent être utilisés qu'en p2 et ne sont *pas* reportés comme le symbole +. Noter aussi qu'il ne doit pas y avoir d'espaces après la partie

^, + ou - de ces symboles -- le nombre doit suivre directement comme dans ^+2.3. Si l'exemple ci-dessus avait été

```
i1 0 .5 100
i . ^+1
i . ^+1
```

le résultat aurait été

```
i1 0 .5 100
i1 1 .5 100
i1 2 .5 100
```

On peut se servir largement de la fonction Carry. Son utilisation, spécialement dans les grandes partitions, peut réduire grandement la frappe au clavier et elle simplifiera les modifications ultérieures.

Il y a des circonstances où l'on ne veut pas que les p-champs "manquants" après le dernier qui a été entré soient implicitement reportés. Par exemple dans un instrument prévu pour prendre un nombre variable de p-champs. A partir de Csound 5.08, on peut empêcher le report implicite des p-champs à la fin d'une instruction *i* en utilisant le symbole ! (appelé le "symbol de non-report"). Le ! doit apparaître à la fin d'une instruction *i* et il ne peut pas être utilisé en *p1*, *p2* ou *p3*, car ces p-champs sont obligatoires. Voici un exemple :

```
i1 0 .5 100
i . +
i . . !
i
```

Cette partition sera interprétée comme ceci

```
i1 0 .5 100
i1 .5 .5 100
i1 1 .5 ; no p4
i1 1.5 .5 ; only p1 to p3 are carried here
```

## Tempo

Cette opération modifie l'information temporelle d'une section de partition selon les directives de l'instruction *t*. L'opération tempo convertit *p2* (et pour les instructions *i*, *p3*) de la valeur originale en pulsations vers des secondes réelles, celles-ci étant les unités temporelles requises par l'orchestre. Après la modification temporelle, les fichiers partitions apparaîtront dans un format lisible par l'orchestre comme ceci :

*i p1 p2*pulsations *p2*secondes *p3*pulsations *p3*secondes *p4 p5* ...

## Sort

Cette routine trie toutes les instructions d'action temporelle chronologiquement selon la valeur de *p2*. Elle place aussi les événements simultanés par ordre de priorité. Chaque fois qu'une instruction *f* et une instruction *i* ont la même valeur en *p2*, l'instruction *f* sera placée en premier. Chaque fois que plusieurs instructions *i* ont la même valeur en *p2*, elles seront triées par ordre croissant de leur valeur en *p1*. Si

elles ont aussi la même valeur en p1, elles seront triées par ordre croissant de leur valeur en p3. Le tri de la partition est effectué par section (voir l'*instruction s*). Ce tri automatique permet d'écrire les instructions de partition dans n'importe quel ordre à l'intérieur d'une section.



### Note

Les opérations Carry, Tempo et Sort sont combinées dans une seule passe en trois phases sur le fichier de partition, pour produire un nouveau fichier dans un format lisible par l'orchestre (voir l'exemple de Tempo). Ce traitement peut être invoqué explicitement par la commande *Scsort*, ou implicitement par Csound qui traite la partition avant d'appeler l'orchestre. Les fichiers en format source et en format lisible par l'orchestre sont encodés en caractères ASCII, et peuvent être consultés ou modifiés dans un éditeur de texte standard. L'utilisateur peut écrire ses propres routines pour modifier les fichiers de partition avant ou après le processus décrit ci-dessus, pourvu que le format final lisible par l'orchestre soit respecté. Les sections de formats différents peuvent être traitées séquentiellement par lots ; et les sections de même format peuvent être réunies pour le tri automatique.

## Instructions de Partition

Les instructions utilisées dans les partitions sont :

- *a* - Avance le temps de la partition d'une quantité spécifiée
- *b* - Réinitialise l'horloge
- *e* - Marque la fin de la dernière section de la partition
- *f* - Appelle une *routine GEN* pour placer des valeurs dans une table de fonction stockée
- *i* - Active un instrument à une date spécifique et pour une certaine durée
- *m* - Positionne une marque nommée dans la partition
- *n* - Répète une section
- *q* - Rend un instrument silencieux
- *r* - Commence une section répétée
- *s* - Marque la fin d'une section
- *t* - Fixe le tempo
- *v* - Permet une modification temporelle variable localement des événements de la partition
- *x* - Ignore le reste de la section courante
- *{* - Commence une boucle imbriquable ne délimitant pas de section
- *}* - Termine une boucle imbriquable ne délimitant pas de section

## Symboles Next-P et Previous-P

A la fin de chacune des opérations *Carry*, *Tempo*, et *Sort*, trois fonctions de partition supplémentaires sont interprétées durant l'écriture du fichier : *next-p*, *previous-p*, et *ramping*.

Les p-champs d'une *instruction* *i* contenant les symboles *npx* ou *ppx* (où *x* est un entier) seront remplacés par la valeur du p-champ approprié de l'instruction *i* suivante (ou de l'instruction *i* précédente) ayant le même p1. Par exemple, le symbole *np7* sera remplacé par la valeur du p7 de la note suivante devant être jouée par le même instrument. Les symboles *np* et *pp* sont récursifs et peuvent référencer d'autres symboles *np* et *pp* qui peuvent en référencer d'autres, etc. Les références doivent se terminer par un nombre réel ou un *symbole ramp*. Il faut éviter les références en boucle fermée. Les symboles *np* et *pp* sont interdits en p1, p2 et p3 (bien qu'ils puissent référencer ces derniers). Les symboles *np* et *pp* peuvent être reportés (Carry). Les références de *np* et de *pp* ne peuvent traverser une limite de Section. Toute référence avant ou arrière à une instruction de note inexistante recevra la valeur zéro.

Par exemple : les instructions

```
i1  0    1    10   np4  pp5
i1  1    1    20
i1  1    1    30
```

se transformeront en

```
i1  0    1    10   20   0
i1  1    1    20   30   20
i1  2    1    30    0   30
```

Les symboles *np* et *pp* peuvent apporter à un instrument une connaissance contextuelle de la partition, ce qui permettra de réaliser un glissando ou un crescendo, par exemple, vers la hauteur ou l'intensité d'un événement futur (qui peut être immédiatement adjacent ou non). A noter que bien que la fonction *Carry* propage *np* et *pp* vers des instructions non triées, l'opération d'interprétation de ces symboles se fait sur une version de la partition résolue temporellement et complètement triée.

## Ramping

Les p-champs d'une *instruction* *i* contenant le symbole < seront remplacés par des valeurs issues de l'interpolation linéaire d'une pente temporelle. Les pentes sont attachées à chaque extrémité au premier nombre réel trouvé dans le même p-champ de notes précédentes et suivantes jouées par le même instrument. Par exemple : les instructions

```
i1  0    1    100
i1  1    1    <
i1  2    1    <
i1  3    1    400
i1  4    1    <
i1  5    1    0
```

se transformeront en

```
i1  0    1    100
i1  1    1    200
i1  2    1    300
i1  3    1    400
i1  4    1    200
i1  5    1    0
```

Les pentes ne peuvent pas traverser une limite de Section. Les pentes ne peuvent pas être attachées à un



symbole *np* ou *pp* (mais elles peuvent être référencées par ceux-ci). Les symboles de pente sont interdits en p1, p2 et p3. Les symboles de pente peuvent être reportés. A noter cependant que, bien que la fonction *Carry* propage les symboles de pente vers des instructions non triées, l'opération d'interprétation de ces symboles se fait sur une version de la partition résolue temporellement et complètement triée. En fait, l'interpolation linéaire temporelle est basé sur le temps de partition résolu, de façon à ce qu'une pente couvrant un groupe de notes *accelerando* reste linéaire par rapport au temps strictement chronologique.

A partir de la version 3.52 de Csound, l'utilisation des symboles ( ou ) donne une pente d'interpolation exponentielle, comme *expon*. L'utilisation du symbole ~ donnera une distribution aléatoire uniforme entre la première et la dernière valeur de la pente. L'utilisation de ces fonctions suit les mêmes règles que la fonction de pente linéaire.

## Macros de Partition

### Description

Les macros sont des substitutions de texte qui sont réalisées dans la partition lors de sa présentation au système. Le système de macro de Csound est très simple, et il utilise les caractères # et \$ pour définir et appeler des macros. C'est un moyen de simplifier l'écriture d'une partition, et une alternative élémentaire aux systèmes de génération de partition complète. Le système de macros de partition est similaire, mais de façon indépendante, au système de macros du langage de l'orchestre.

*#define* NOM -- définit une macro simple. Le nom de la macro doit commencer par une lettre et peut être une combinaison de lettres et de nombres. La casse est significative. Cette forme est restrictive dans le sens que les noms de variable sont fixes. On peut obtenir plus de souplesse au moyen d'une macro avec arguments, décrite ci-dessous.

*#define* NOM(a' b' c') -- définit une macro avec arguments. On peut l'utiliser dans des situations plus complexes. Le nom de la macro doit commencer par une lettre et peut être suivi par une combinaison de lettres et de chiffres. Dans le texte de substitution, les arguments sont remplacés par la forme : \$A. En fait, les arguments sont implémentés comme des macros simples. Il peut y avoir jusqu'à 5 arguments, et leur nom peut être n'importe quel choix de lettres. Rappelez-vous que la casse est significative dans les noms de macro.

*\$NOM.* -- appelle une macro définie. Pour appeler une macro, on utilise son nom précédé d'un caractère \$. Le nom se termine par le premier caractère qui n'est ni une lettre ni un chiffre. Si on ne veut pas terminer le nom par un espace, on peut utiliser un point qui sera ignoré. La chaîne, *\$NOM.*, est remplacée par le texte de substitution de la définition. Le texte de substitution peut aussi contenir des appels de macro.

*#undef* NOM -- rend un nom de macro indéfini. Si l'on a plus besoin d'une macro, on peut la rendre indéfinie avec *#undef* NOM.

### Syntaxe

```
#define NOM # texte de substitution #
```

```
#define NOM(a' b' c') # texte de substitution #
```

```
$NOM.
```

```
#undef NOM
```

## Initialisation

*# texte de substitution #* -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est délimité par des caractères #, ce qui permet d'éviter l'insertion de caractères supplémentaires par inadvertance.

## Exécution

Il faut prendre quelques précautions avec les macros de substitution de texte, car elle peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

**Une Autre Utilisation des Macros.** Lorsque l'on écrit une partition complexe, on oublie parfois trop facilement à quoi les différents numéros d'instruments font référence. On peut utiliser des macros pour nommer ces nombres. Par exemple

```
#define Flute #i1#
#define Whoop #i2#

$Flute. 0 10 4000 440
$Whoop. 5 1
```

## Exemples

### Exemple 1. Macro Simple

Une note a un ensemble de p-champs qui sont répétés :

```
#define ARGS # 1.01 2.33 138#
i1 0 1 8.00 1000 $ARGS
i1 0 1 8.01 1500 $ARGS
i1 0 1 8.02 1200 $ARGS
i1 0 1 8.03 1000 $ARGS
```

Ce sera développé avant le tri en :

```
i1 0 1 8.00 1000 1.01 2.33 138
i1 0 1 8.01 1500 1.01 2.33 138
i1 0 1 8.02 1200 1.01 2.33 138
i1 0 1 8.03 1000 1.01 2.33 138
```

On économise ainsi de la frappe au clavier, et les révisions sont plus faciles. Avec deux ensembles de p-champs on pourrait avoir une seconde macro (il n'y pas de réelle limite au nombre de macros que l'on peut définir).

```
#define ARGS1 # 1.01 2.33 138#
#define ARGS2 # 1.41 10.33 1.00#
i1 0 1 8.00 1000 $ARGS1
i1 0 1 8.01 1500 $ARGS2
i1 0 1 8.02 1200 $ARGS1
i1 0 1 8.03 1000 $ARGS2
```

## Exemple 2. Macros avec arguments

```
#define ARG(A) # 2.345 1.03 $A 234.9#  
i1 0 1 8.00 1000 $ARG(2.0)  
i1 + 1 8.01 1200 $ARG(3.0)
```

qui se développe en

```
i1 0 1 8.00 1000 2.345 1.03 2.0 234.9  
i1 + 1 8.01 1200 2.345 1.03 3.0 234.9
```

## Crédits

Auteur : John ffitich

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

# Partition dans Plusieurs Fichiers

## Description

Disposer la partition dans plusieurs fichiers.

## Syntaxe

```
#include "nomfichier"
```

## Exécution

Il est parfois commode de disposer la partition dans plusieurs fichiers. On peut le faire en utilisant *#include* qui fait partie du système de macro. Par une ligne contenant le texte

```
#include "nomfichier"
```

où le caractère " peut être remplacé par n'importe quel caractère adéquat. Pour la plupart des usages, le symbole des guillemets sera probablement le plus adapté. Le nom de fichier peut comprendre un nom de chemin complet.

On prend en entrée le contenu du fichier nommé, puis on revient à l'entrée précédente. La profondeur des fichiers inclus et des macros est actuellement limitée à 20.

On peut utiliser *#include* pour définir un ensemble de macros qui font partie du style du compositeur. On peut aussi l'utiliser pour répéter des sections.

```
s
#include :section1:
;; Répéter ceci
s
#include :section1:
```

Pour d'autres méthodes de répétition, utiliser l'instruction *r*, l'instruction *m*, et l'instruction *n*.

## Crédits

Auteur : John ffitich

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

Merci à Luis Jure d'avoir relevé la syntaxe incorrecte dans l'instruction d'inclusion de fichiers.

## Evaluation des Expressions

Dans les anciennes versions de Csound les nombres présents dans une partition étaient utilisés tels quels. Dans certains cas, une évaluation simple serait plus facile. Ce besoin est accru s'il y a des macros. Pour y arriver, on a introduit la syntaxe des expressions arithmétiques entre crochets [ ]. On peut utiliser des expressions avec les opérations +, -, \*, /, % ("modulo"), et ^ ("élévation à une puissance"), les groupements se faisant par parenthèses ( ). Les signes unaires plus et moins sont aussi supportés. Les expressions peuvent inclure des nombres et, naturellement, des macros dont la valeur est une chaîne numérique ou arithmétique. Tous les calculs sont faits en nombres en virgule flottante. Les règles de précedence usuelles sont suivies lors de l'évaluation : les expressions entre parenthèse ( ) sont évaluées en premier et ^ est évalué avant \*, /, et % qui sont évalués avant + et -.

En plus des opérations arithmétiques, les opérateurs logiques bit à bit suivants sont aussi disponibles : & (ET), | (OU), et # (OU exclusif). Ces opérateurs arrondissent leurs opérandes à l'entier (long) le plus proche avant l'évaluation. Les opérateurs logiques ont la même précedence que les opérateurs arithmétiques \*, /, et %.

On a ajouté dans la version 3.56 de Csound @x (la première puissance de deux supérieure ou égale à x) et @@x (la première puissance de deux plus un supérieure ou égale à x).

Finalement, on peut utiliser le symbole tilde ~ dans une expression chaque fois qu'un nombre est permis. Chaque ~ sera remplacé par un nombre aléatoire compris entre zéro (0) et un (1).

## Exemple

```
r3  CNT
i1  0  [0.3*$CNT.]
i1  +  [( $CNT./3)+0.2]
e
```

Comme les trois copies de la section comprennent la macro \$CNT. avec les valeurs successives 1, 2 et 3, le développement est

```
s
i1 0 0.3
i1 0.3 0.533333
s
i1 0 0.6
i1 0.6 0.866667
s
i1 0 0.9
i1 0.9 1.2
e
```

C'est une forme extrême, mais on peut aussi utiliser le système d'évaluation pour répéter des sections avec des différences subtiles.

Voici quelques exemples simples de chaque opérateur :

```
i1 0 1 [ 110 + 220 ] ; evaluates to 330
i1 + . [ 330 - 55 ] ; 275
i1 + . [ 44 * 10 ] ; 440
i1 + . [ 1100 / 2 ] ; 550
i1 + . [ 5 ^ 4 ] ; 625
i1 + . [ 5660 % 1000 ] ; 660
i1 + . [ 110 & 220 ] ; 76
i1 + . [ 110 | 220 ] ; 254
i1 + . [ 110 # 220 ] ; 178
i1 + . [ ~ ] ; random between 0-1
i1 + . [ ~ * 4 + 1 ] ; random between 1-5
i1 + . [ ~ * 95 + 5 ] ; random between 5-100

i1 + . [ 8 / 2 * 3 ] ; 12
i1 + . [ 4 + 3 - 2 + 1 ] ; 6
i1 + . [ 4 + 3 * 2 + 1 ] ; 11
i1 + . [ (4 + 3)*(2 + 1) ] ; 21

i1 + . [ 2 * 2 & 3 ] ; 4
i1 + . [ 3 & 2 * 2 ] ; 0
i1 + . [ 4 | 3 * 3 ] ; 13
```

## Crédits

Auteur : John ffitich

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

## Chaînes de caractères dans les p-champs

On peut passer une chaîne de caractères dans un p-champ au lieu d'un nombre, comme ceci :

```
i 1 0 10 "A4"
```

Cette chaîne de caractères peut être reçue par l'instrument et traitée par les *opcodes de chaîne de caractères*.



### Note

Actuellement un seul p-champ peut contenir une chaîne de caractères (c-à-d qu'on n'autorise pas plus d'une chaîne de caractères par ligne). On peut contourner ceci en utilisant *strset* et *strget*.

---

# Frontaux

Voici une liste (non exhaustive) des frontaux disponibles pour Csound.

## Csound5GUI

Csound5GUI est une interface utilisateur graphique (GUI) multi plates-formes, polyvalente qui fait partie de la distribution standard de Csound. Elle implémente la plupart des options de configuration de Csound.

## CSDplayer

C'est un simple programme java pour jouer des fichiers csd. Il est inclus dans la distribution standard.

## Winsound

Egalement présent dans l'arborescence principale de Csound (bien qu'absent de certaines distributions), Winsound est un portage multi plates-formes en FLTK du frontal original de Barry Vercoe pour Csound.

## WinXoundPro

Un frontal commmode pour windows avec coloration syntaxique. On peut l'obtenir à WinX-sound Front Page [<http://www.ibiart.it/winxound/index.html>].

## Csound Editor

Un frontal commmode pour windows avec coloration syntaxique. On peut l'obtenir à Flavio Tordini's Home Page [<http://flavio.tordini.org/csound-editor/>].

## MacCsound

Plus qu'un frontal pour le Mac à MacCsound Page [<http://www.csounds.com/matt/MacCsound/>].

## Cabel

Cabel est une interface utilisateur graphique pour construire des instruments csound en interconnectant des modules similaires aux modules des synthétiseurs. Multi plates-formes, écrit en Python. A <http://cabel.sourceforge.net/>.

## Blue

Frontal orienté composition, écrit en Java. Son interface ressemble beaucoup à un multipiste numérique, mais en diffère en intégrant des axes temporels dans des axes temporels (polyObjects). Cela permet une organisation compositionnelle qui me semble très intuitive, instructive et flexible. Téléchargeable à : Blue Home Page [<http://csounds.com/stevenyi/blue/>].

# CsoundAC

## Programmation Python

Vous pouvez utiliser CsoundAC comme un module d'extension de Python. Vous pouvez faire cela dans un interpréteur Python standard tel que la ligne de commande Python ou le Idle Python GUI.

Pour utiliser CsoundAC dans un interpréteur Python standard, importez CsoundAC.

```
import CsoundAC
```

Le module CsoundAC crée automatiquement une instance de CppSound nommée `csound`, qui fournit une interface orientée objet à l'API de Csound. Dans un interpréteur Python standard, vous pouvez charger un fichier Csound `.csd` et l'exécuter de cette manière :

```
C:\Documents and Settings\mkg>python
Python 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1200 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import CsoundAC
>>> csound.load("c:/projects/csound5/examples/trapped.csd")
1
>>> csound.exportForPerformance()
1
>>> csound.perform()
BEGAN CppSound::perform(5, 988ee0)...
BEGAN CppSound::compile(5, 988ee0)...
Using default language
0dBFS level = 32767.0
Csound version 5.00 beta (float samples) Jun  7 2004
libsndfile-1.0.10pre6
orchname: temp.orc
scorename: temp.sco
orch compiler:
398 lines read
instr 1
instr 2
instr 3
instr 4
instr 5
instr 6
instr 7
instr 8
instr 9
instr 10
instr 11
instr 12
instr 13
instr 98
instr 99
sorting score ...
... done
Csound version 5.00 beta (float samples) Jun  6 2004
displays suppressed
0dBFS level = 32767.0
orch now loaded
audio buffered in 16384 sample-frame blocks
SFDIR undefined. using current directory
writing 131072-byte blks of shorts to test.wav
WAV
SECTION 1:
ENDED CppSound::compile.
ftable 1:
ftable 2:
ftable 3:
ftable 4:
ftable 5:
ftable 6:
ftable 7:
ftable 8:
ftable 9:
ftable 10:
ftable 11:
```



```

ftable 12:
ftable 13:
ftable 14:
ftable 15:
ftable 16:
ftable 17:
ftable 18:
ftable 19:
ftable 20:
ftable 21:
ftable 22:
new alloc for instr 1:
B 0.000 .. 1.000 T 1.000 TT 1.000 M: 32.7 0.0
new alloc for instr 1:
B 1.000 .. 3.600 T 3.600 TT 3.600 M: 207.6 0.1
...

B 93.940 .. 94.418 T 98.799 TT281.799 M: 477.6 85.0
B 94.418 ..100.000 T107.172 TT290.172 M: 118.9 11.5
end of section 4 sect peak amps: 25950.8 26877.4
inactive allocs returned to freespace
end of score. overall amps: 32204.8 31469.6
overall samples out of range: 0 0
0 errors in performance
782 131072-byte soundblks of shorts written to test.wav WAV
Elapsed time = 13.469000 seconds.
ENDED CppSound::perform.
1
>>>

```

Le script `koch.py` montre comment utiliser Python pour faire une composition algorithmique pour Csound. Vous pouvez utiliser des chaînes de caractères littérales à triples guillemets pour incorporer vos fichiers Csound directement dans votre script, et les assigner à Csound :

```

csound.setOrchestra(''sr = 44100
kr = 441
ksmps = 100
nchnls = 2
0dbfs = .1
instr 1,2,3,4,5 ; FluidSynth General MID
I; INITIALIZATION
; Channel, bank, and program determine the preset, that is, the actual sound.
ichannel = p1
iprogram = p6
ikey = p4
ivelocity = p5 + 12
ijunk6 = p6
ijunk7 = p7
; AUDIO
istatus = 144;
print iprogram, istatus, ichannel, ikey, ivelocityleft, aright
fluid "c:/projects/csound5/samples/VintageDreamsWaves-v2.sf2", \
iprogram, istatus, ichannel, ikey, ivelocity, 1
outs aleft, arightendin'')
csound.setCommand("csound --opcode-lib=c:/projects/csound5/fluid.dll \
-RWdfo ./koch.wav ./temp.orc ./temp.sco")
csound.exportForPerformance()
csound.perform()

```

## CsoundVST

CsoundVST est un frontal multi-fonction pour Csound, basé sur l'API de Csound. CsoundVST s'exécute comme une interface utilisateur graphique autonome pour Csound, et il s'exécute aussi comme un instrument VST ou un plugin d'effet dans des hôtes VST tels que Cubase avec la même interface utilisateur. CsoundVST fait partie de l'arbre principal des sources de Csound, mais il n'est pas inclus dans les distributions standard à cause des limitations de la license du SDK VST de Steinberg.

## Utilisation autonome

Pour lancer CsoundVST comme frontal autonome pour Csound, exécutez CsoundVST. Au démarrage du programme, vous verrez une interface graphique utilisateur avec une rangée de boutons en haut. Cliquez sur le bouton *Open...* pour charger un fichier `.csd`. Vous pouvez aussi cliquer sur le bouton *Open...* et charger un fichier `.orc`, cliquez ensuite sur le bouton *Import...* pour ajouter un fichier `.sco`. Vous pouvez éditer la commande de Csound, le fichier orchestre, ou le fichier partition dans les onglets respectifs de l'interface utilisateur. Quand tout est prêt, cliquez sur le bouton *Perform* pour lancer Csound. Vous pouvez arrêter une exécution à n'importe quel moment en cliquant sur le bouton *Stop*.

## Plugin VST

Les instructions suivantes sont pour Cubase 4.0. Des procédures à peu près similaires seraient utilisées dans d'autres programmes hôtes.

Utilisez le menu *Devices*, la boîte de dialogue *Plug-In Information*, l'onglet *VST Plug-Ins*, la boîte de dialogue *VST 2.x Plug-in Paths*, le bouton *Add* pour ajouter votre répertoire `csound/bin` au chemin des plugins de Cubase. Vous pouvez avoir plusieurs répertoires séparés par des points-virgules. Sélectionnez ensuite le chemin de CsoundVST et cliquez sur le bouton *Set as Shared Folder*

Quittez Cubase, et redémarrez-le.

Utilisez le menu *File*, la boîte de dialogue *New Project* pour créer un nouveau morceau (song).

Utilisez le menu *Project*, le sous-menu *Add Track*, pour ajouter une nouvelle piste MIDI.

Utilisez l'outil crayon pour dessiner un *Part* de quelques mesures sur la piste. Ecrivez un peu de musique dans le *Part* à l'aide de l'éditeur *Event* ou de l'éditeur *Score*.

Utilisez le menu *Devices* (ou la touche F11) pour ouvrir la boîte de dialogue *VST Instruments*.

Cliquez sur une des étiquettes *No VST Instrument*, et sélectionnez *CsoundVST* dans la liste qui apparaît.

Cliquez sur le bouton *e* (pour edit) pour ouvrir la boîte de dialogue de *CsoundVST*.

Sur la page des Réglages, cochez la case *Instrument* dans le groupe *VST Plugin*, et la case *Classic* dans le groupe *Csound performance mode*. Cliquez ensuite sur le bouton *Apply*.

Cliquez sur le bouton *Open* pour faire apparaître la boîte de dialogue de sélection de fichier. Naviguez vers un répertoire contenant un fichier `csd` Csound adéquat pour une exécution MIDI, tel que `csound/examples/CsoundVST.csd`. Cliquez sur le bouton *OK* pour charger le fichier. Vous pouvez aussi ouvrir et importer des fichiers `.orc` et `.sco` adéquats comme décrit ci-dessus.

Dans tous les cas, la ligne de commande dans le champ texte *Classic Csound command line* doit spécifier `++rtmidi=null -M0`, et devrait ressembler à ceci :

```
csound -f -h ++rtmidi=null -M0 -d -n -m7 --midi-key-oct=4 --midi-velocity=5 temp.orc temp.sco
```

Cliquez sur le bouton on/off de la boîte de dialogue *VST Instruments* pour l'allumer. Ceci devrait compiler l'orchestre Csound.

Dans l'*Inspecteur de Piste de Cubase*, cliquez sur l'étiquette *out: Not Assigned* et sélectionnez *CsoundVST* dans la liste qui apparaît.

Sur la règle en haut de la fenêtre *Arrangement*, sélectionnez le point de fin de boucle et tirez-le jusqu'à la fin de votre part, cliquez ensuite sur le bouton *loop* pour activer la mise en boucle.

Cliquez sur le bouton *play* de la barre de *Transport*. Vous devriez entendre votre musique jouée par CsoundVST.

Essayez d'assigner votre piste à différents canaux ; un instrument Csound différent jouera chaque canal.

Quand vous sauvegardez votre song, votre orchestre Csound sera sauvegardé comme une partie du song et rechargé quand vous rechargerez le song.

Vous pouvez cliquer sur l'onglet *Orchestra* et éditer vos instruments Csound pendant que CsoundVST est en train de jouer. Pour entendre vos changements, il suffit de cliquer sur le bouton CsoundVST *Perform* pour recompiler l'orchestre.

Vous pouvez assigner jusqu'à 16 canaux à un seul plugin CsoundVST.

---

# TclCsound

TclCsound fut introduit pour fournir une interface simple de scripting à Csound. Tcl est un langage simple aisément extensible et qui facilite des opérations comme l'accès aux fichiers et la mise en réseau sous TCP. Avec son composant Tk, il peut aussi gérer une interface graphique pilotée par événements. TclCsound donne trois "points de contact" avec Tcl :

1. un interpréteur tcl connaissant csound (cstclsh)
2. un shell de fenêtrage connaissant csound (cswish)
3. un module de commandes csound pour Tcl/Tk (bibliothèque dynamique tclcsound)

## L'interpréteur Tcl : cstclsh

Avec cstclsh, on peut contrôler de manière interactive une exécution csound. La commande démarre un shell interactif, qui maintient une instance de Csound. On peut ensuite utiliser plusieurs commandes pour la contrôler. Par exemple, la commande suivante peut compiler du code csound et le charger en mémoire, prêt à être exécuter :

```
csCompile -odac orchestre partition -m0
```

Ceci fait, on peut démarrer l'exécution de deux manières : avec csPlay ou avec csPerform. La commande

```
csPlay
```

démarrera l'exécution Csound dans un thread séparé et retournera à l'invite de cstclsh. On peut utiliser ensuite plusieurs commandes pour contrôler Csound. Par exemple,

```
csPause
```

suspendra l'exécution ; et

```
csRewind
```

reviendra au début de la liste de notes. On peut utiliser les commandes csNote, csTable et csEvent pour ajouter des événements de partition pendant l'exécution, à la volée. La commande csPerform, à l'inverse de csPlay, ne lancera pas un thread séparé, mais démarrera Csound dans le même thread, ne retournant que quand l'exécution est finie. Il existe une variété d'autres commandes, donnant un contrôle total de Csound.

## Cswish: le shell de fenêtrage

Avec Cswish, on peut utiliser des commandes et des widgets Tk pour se doter d'une interface graphique avec gestion d'événements. Comme pour cstclsh, le lancement de la commande cswish ouvre aussi un shell interactif. Par exemple, on peut utiliser les commandes suivantes pour créer un panneau de transport pour Csound :

```
frame .fr
button .fr.play -text play -command csPlay
button .fr.pause -text pause -command csPause
button .fr.rew -text rew -command csRewind
pack .fr .fr.play .fr.pause .fr.rew
```

De même, on peut lier des touches à des commandes afin d'utiliser le clavier de l'ordinateur pour jouer avec Csound.

Les commandes de contrôle de canal fournies par TclCsound sont particulièrement utiles. Par exemple, on peut enregistrer des canaux d'E/S nommés avec TclCsound et les utiliser avec les opcodes invalve et outvalue. De plus, l'API de Csound fournit aussi un bus logiciel complet pour les canaux audio, de contrôle et de chaînes. Dans TclCsound, on peut accéder aux canaux du bus de contrôle et de chaînes (le bus audio n'est pas implémenté, car Tcl n'est pas capable de traiter ce genre de données). Avec ces commandes de TclCsound, on peut connecter facilement des widgets Tk aux paramètres de synthèse.

## Un serveur Csound

Dans Tcl, il est très simple de configurer des connexions réseau TCP. On peut construire un serveur csound avec quelques lignes de code. Celui-ci peut accepter des connexions depuis la machine locale ou depuis des clients distants. Non seulement les clients Tcl/Tk peuvent lui envoyer des commandes, mais des connexions TCP peuvent être établies depuis un autre logiciel, comme par exemple, Pure Data (PD). On montre ci-dessous un script Tcl qui peut être lancé dans l'interpréteur standard tclsh. Il utilise le module Tclcsound, une bibliothèque dynamique qui ajoute les commandes de l'API de Csound à Tcl.

```
# load tclcsound.so
#(OSX: tclcsound.dylib, Windows: tclcsound.dll)
load tclcsound.so Tclcsound
set forever 0

# This arranges for commands to be evaluated
proc ChanEval { chan client } {
  if { [catch { set rtn [eval [gets $chan]] } err] } {
    puts "Error: $err"
  } else {
    puts $client $rtn
    flush $client
  }
}

# this arranges for connections to be made

proc NewChan { chan host port } {
  puts "Csound server: connected to $host on port $port ($chan)"
  fileevent $chan readable [list ChanEval $chan $host]
}

# this sets up a server to listen for
# connections

set server [socket -server NewChan 40001]
set sinfo [fconfigure $server -sockname]
puts "Csound server: ready for connections on port [lindex $sinfo 2]"
vwait forever
```

Lorsque le serveur est actif, il est alors possible de configurer des clients pour contrôler le serveur Csound. On peut lancer de tels clients depuis des interpréteurs Tcl/Tk standard, car ils n'évaluent pas eux-mêmes les commandes Csound. Voici un exemple de connexions client à un serveur Csound au moyen de Tcl :

```
# connect to server
set sock [socket localhost 40001]

# compile Csound code
puts $sock "csCompile -odac orchestra score"
```

```
flush $sock

# start performance
puts $sock "csPlay"
flush $sock

# stop performance
puts $sock "csStop"
flush $sock
```

Comme il est mentionné ci-dessus, on peut configurer des clients utilisant d'autres systèmes logiciels, tels que PD. De tels clients n'ont besoin que de se connecter au serveur (au moyen d'un objet netsend) et de lui envoyer des messages. Le premier élément de chaque message est une commande. D'autres éléments facultatifs peuvent y être ajoutés comme arguments de cette commande.

## Un Environnement de Scripting

Avec TclCsound, on peut transformer le populaire éditeur de texte emacs en environnement de scripting et d'exécution de Csound. Lorsqu'il est en mode Tcl, l'éditeur permet d'évaluer des expressions Tcl par sélection et utilisation d'une simple séquence d'échappement (Ctrl-C Ctrl-X). Grâce à cela, on peut éditer et exécuter du code Csound et Tcl/Tk de façon intégrée

Dans Tcl il est possible d'écrire des fichiers de partition et d'orchestre qui peuvent être sauvegardés, compilés et exécutés par le même script, sous l'environnement emacs. L'exemple suivant montre un script Tcl qui construit un instrument csound et lance ensuite une exécution de csound. Il crée 10 oscillateurs en parallèle légèrement désaccordés, ce qui génère des sons semblables à ceux que l'on trouve dans *Inharmonique* de Risset.

```
load tclcsound.so TclCsound

# set up some intermediary files

set orcfiler "tcl.orc"
set scofile "tcl.sco"
set orc [open $orcfiler w]
set sco [open $scofile w]

# This Tcl procedure builds an instrument
proc MakeIns { no code } {
    global orc sco
    puts $orc "instr $no"
    puts $orc $code
    puts $orc "endin"
}

# Here is the instrument code
append ins "asum init 0 \n"
append ins "ifreq = p5 \n"
append ins "iamp = p4 \n"

for { set i 0 } { $i < 10 } { incr i } {
    append ins "a$i oscili iamp,
ifreq+ifreq*[expr $i * 0.002], 1\n"
}

for { set i 0 } { $i < 10 } { incr i } {
    if { $i } {
        append ins " + a$i"
    } else {
```

```
append ins "asum = a$i "
}

append ins "\nk1 linen 1, 0.01, p3, 0.1 \n"
append ins "out asum*k1"

# build the instrument and a dummy score

MakeIns 1 $ins
puts $sco "f0 10"
close $orc
close $sco

# compile
csCompile $orcfile $scofile -odac -d -m0

# set a wavetable
csTable 1 0 16384 10 1 .5 .25 .2 .17 .15 .12 .1

# send in a sequence of events and perform it
for {set i 0} { $i < 60 } { incr i } {
  csNote 1 [expr $i * 0.1] .5 \
    [expr ($i * 10) + 500] [expr 100 + $i * 10]
}
csPerform

# it is possible to run it interactively as
# well
csNote 1 0 10 1000 200
csPlay
```

De telles facilités comme celles fournies par emacs permettent d'émuler un environnement assez proche de ce qu'on trouve dans les soi-disant "systèmes de synthèse modernes", tels que SuperCollider (SC). En fait, on peut exécuter Csound dans une configuration client-serveur, ce qui est une des fonctionnalités de SC3. Csound a l'avantage majeur de fournir trois ou quatre fois plus de générateurs unitaires que ce qu'on trouve dans ce langage (de même qu'il fournit une approche du traitement du signal à un plus bas niveau, en fait ce ne sont là que quelques-uns des avantages de Csound).

## TclCsound comme encapsuleur de langage

On peut utiliser TclCsound à un niveau légèrement plus bas, car beaucoup des fonctions de l'API C ont été encapsulées dans des commandes Tcl. Par exemple, il est possible de créer un frontal "classique" pour csound en ligne de commande complètement écrit en Tcl. Le script suivant le démontre :

```
#!/usr/local/bin/cstclsh

set result 1
csCompileList $argv
while { $result != 0 } {
  set result csPerformKsmps
}
```

## Référence des Commandes de TclCsound

Commandes de contrôle de l'exécution :

**csCompile [ligne de commande csound]** : compile un orc/sco/csd + des options

**csCompileList arglist** : compile un orc/sco/csd + des options, donnés comme une liste Tcl 'arglist'

**csPerform** : joue la partition, retournant à la fin

**csPerformKsmpls** : exécute un bloc de ksmpls échantillons audio, puis retourne

**csPerformBuffer** : exécute un bloc d'échantillons audio de la taille d'un tampon, puis retourne

**csPlay** : démarre une exécution asynchrone dans un thread séparé, retournant immédiatement

**csPause** : suspend la reproduction

**csStop** : arrête l'exécution et réinitialise csound

**csRewind** : repositionne la partition au début

**csOffset secs** : décale le point de reproduction dans la partition de 'secs' secondes

**csGetoffset** : retourne le point de décalage dans la partition en secondes

**csGetScoreTime** : retourne le temps de la partition en secondes

Commandes d'évènements :

**csNote [p-champs]** : envoie un évènement dans une instruction i

**csTable [p-champs]** : envoie un évènement dans une instruction f

**csEvent opcode [p-champs]** : envoie un évènement de partition défini par 'opcode' plus les p-champs

**csNoteList arglist** : envoie un évènement dans une instruction i avec les p-champs dans une liste Tcl 'arglist'

**csTableList arglist** : envoie un évènement dans une instruction f avec les p-champs dans une liste Tcl 'arglist'

**csEventList arglist** : envoie un évènement de partition défini par 'opcode' avec les p-champs dans une liste Tcl 'arglist'

Commandes de canal de contrôle et de chaîne, invaluel, outvalue, pvsin, pvsout :

**csInChannel nom** : enregistre un canal csound invaluel

**csOutChannel nom** : enregistre un canal csound outvalue et crée la variable tcl globale 'nom'

**csInValue canal valeur** : fixe une valeur sur un canal csound invaluel

**csOutValue canal** : retourne la valeur d'un canal csound outvalue

**csPvsIn number [size olaps wsize wtype]** : enregistre un canal du bus d'entrée pvs, initialisant optionnellement les valeurs de fsig à une taille de tfr de 'size' (par défaut : 1024), une taille de chevauchement de 'olaps' (par défaut : size/4), une taille de fenêtre de 'wsize' (par défaut : size) et le type de fenêtre à 'wtype' (par défaut : 1, fenêtre de Hanning, voir la page de manuel pour pvsanal). Fonctionne avec l'opcode pvsin (seulement le format PVS\_AMP\_FREQ).

**csPvsOut number [size olaps wsize wtype]** : enregistre un canal du bus de sortie pvs. Fonctionne avec



l'opcode pvsout (seulement le format PVS\_AMP\_FREQ).

**csPvsInSet channel bin amp freq** : fixe l'amplitude et la fréquence d'un bin du canal d'entrée pvs 'channel'.

**csPvsOutGet channel bin [isFreq]** : retourne l'amplitude ou la fréquence d'un bin du canal de sortie pvs 'channel'. L'argument optionnel 'isFreq' (par défaut : 0) contrôle si la valeur retournée est l'amplitude du bin (0) ou sa fréquence (1).

**csSetControlChannel channel value** : fixe la valeur du canal de contrôle 'channel', le créant s'il n'existe pas.

**csGetControlChannel channel** : retourne la valeur du canal de contrôle 'channel', le créant s'il n'existe pas.

**csSetStringChannel channel string** : fixe la chaîne dans le canal 'channel', le créant s'il n'existe pas.

**csGetStringChannel channel** : retourne la chaîne qui est dans le canal 'channel', le créant s'il n'existe pas.

Commandes de message :

**csMessageOutput var** : ajoute tous les messages csound à la variable tcl 'var'.

Commandes de table :

**csGetTableSize ftn** : retourne la taille de la table de fonction ftn (-1 si elle n'existe pas).

**csSetTable ftn index value** : fixe la valeur de la position 'index' dans la table de fonction 'ftn' à 'value'.

**csGetTable ftn index** : retourne la valeur de la position 'index' dans la table de fonction 'ftn'.

Commandes de variable d'environnement :

**csOpcodedir opcodedir** : fixe le répertoire des opcode.

**csSetenv envvar value** : fixe la valeur d'une variable d'environnement (par exemple SFDIR, SADIR).

---

# Construire Csound

Csound est devenu un projet complexe et peut impliquer plusieurs dépendances. A moins d'être un développeur de Csound ou d'avoir besoin d'écrire des plugins pour Csound, il vaut mieux utiliser une des distributions pré-compilées de <http://www.sourceforge.net/projects/csound>. Cependant, la construction à partir des sources est sans doute la meilleure option sous GNU/Linux.

Cette section met l'accent sur le système principal de construction de Csound 5, qui utilise SCons [<http://www.scons.org>], un programme Python qui remplace *make* pour la configuration et la construction multi plates-formes.

Lorsque l'on construit Csound à partir des sources plutôt que d'utiliser un paquetage précompilé, il faut d'abord télécharger les sources d'une publication de Csound à partir de <http://www.sourceforge.net/projects/csound>. Les paquetages source ont une extension zip ou tar.gz.

Le code source de Csound le plus récent (potentiellement instable) est également disponible par Concurrent Versions System (CVS). Il est probable (si vous êtes sous Mac OS X ou Linux) que CVS est déjà installé sur votre machine. Si ce n'est pas le cas, il peut être téléchargé à partir de (<http://www.cvshome.org>). Il y a de nombreux frontaux graphiques pour cvs, mais il est facile de télécharger les sources au moyen de la version en ligne de commande.

La page d'accueil du CVS de Csound se trouve ici : [http://sourceforge.net/cvs/?group\\_id=81968](http://sourceforge.net/cvs/?group_id=81968) On peut trouver de l'information sur la manière d'accéder à l'entrepôt CVS de Csound dans le document de SourceForge <http://sourceforge.net/docs/E04/> Pour télécharger les sources de Csound avec CVS, exécutez les commandes suivantes (à partir d'un terminal ou d'un shell DOS) :

```
cvs -d:pserver:anonymous@csound.cvs.sourceforge.net:/cvsroot/csound login
cvs -z3 -d:pserver:anonymous@csound.cvs.sourceforge.net:/cvsroot/csound co -P csound5
```

Pour mettre à jour les sources de Csound5 que vous auriez déjà dans votre répertoire csound5, allez dans ce répertoire et tapez :

```
cvs -z3 update -d
```

Pour mettre à jour un seul fichier, allez dans le répertoire des sources et tapez :

```
cvs -z3 update filename
```

## Conditions nécessaires pour construire Csound 5 sur toutes les plates-formes

- Installer libsndfile version 1.0.13 ou ultérieure depuis [www.mega-nerd.com/libsndfile](http://www.mega-nerd.com/libsndfile) [<http://www.mega-nerd.com/libsndfile>].
- Installer Python depuis [www.python.org](http://www.python.org) [<http://www.python.org>]. Dans la plupart des cas il vaut mieux installer la version stable la plus récente. Scons a besoin de Python pour fonctionner.

- Installer le système de construction SCons depuis [www.scons.org](http://www.scons.org) [<http://www.scons.org>].

Ce sont les conditions minimales pour une construction, mais csound a beaucoup de composants optionnels qui améliorent ses fonctionnalités et qui ajoutent des opcodes pouvant avoir besoin de bibliothèques supplémentaires.

## Configurations optionnelles (TOUTES les plates-formes)

Dans la plupart des cas, il vaut mieux installer les versions stables les plus récentes des bibliothèques optionnelles.

- L'audio en temps réel peut utiliser la bibliothèque multi plates-formes PortAudio (version principale ou branche devel-19) depuis [www.portaudio.com/usingcvs.html](http://www.portaudio.com/usingcvs.html) [<http://www.portaudio.com/usingcvs.html>]. A noter que la version stable 18 ne fonctionnera pas. Csound peut aussi utiliser plusieurs APIS spécifiques aux plates-formes telles que ALSA, JACK, CoreAudio et la bibliothèque multimedia de Windows. Voir les notes de chaque plate-forme pour les détails.
- Le MIDI en temps réel peut utiliser la bibliothèque multi plates-formes PortMidi depuis [www.cs.cmu.edu/~music/portmusic](http://www.cs.cmu.edu/~music/portmusic) [<http://www.cs.cmu.edu/~music/portmusic>]
- Pour les widgets d'interface graphique, installer FLTK 1.1 ou 1.3 depuis [www.fltk.org](http://www.fltk.org) [<http://www.fltk.org>]. Il faut configurer et construire FLTK avec `--enable-shared --enable-threads`.
- Pour générer les interfaces Python et Java, installer le Software Interface and Wrapper Generator (SWIG) depuis <http://www.swig.org>.
- *CsoundAC* nécessite FLTK et les bibliothèques de template C++ boost pour les nombres aléatoires et l'algèbre linéaire, depuis <http://www.boost.org>. *CsoundAC* nécessite au moins la version 1.32.1.
- Les opcodes fluid nécessitent la bibliothèque Fluidsynth depuis <http://savannah.nongnu.org/download/fluid>.
- Les opcodes OSC nécessitent la dernière version de la bibliothèque liblo depuis <http://plugin.org.uk/liblo>. Sous Windows, liblo nécessite une version Windows de la bibliothèque de processus légers POSIX (pthreads) qui est disponible à <http://sourceware.org/pthreads-win32> ; copier libpthreadGC2.a vers libpthread.a. On peut aussi avoir besoin de la dernière version d'autoconf de MinGW.
- Les opcodes STK nécessitent le code source de STK depuis <http://ccrma.stanford.edu/software/stk>, à copier dans `csound5/Opcodes/stk`.
- Les opcodes de Loris nécessitent le code source de Loris depuis <http://sourceforge.net/projects/loris>, à copier dans `csound5/Opcodes/Loris`.

## Windows

On a besoin des éléments suivants pour la construction sous Windows (on peut trouver des instructions plus complètes pour la construction sous Windows dans le document `csound-build.tex` (`csound-build.pdf`)) :

- Installer un compilateur comme gcc ou Microsoft Visual Studio (le compilateur C++ d'Intel est également supporté). Si l'on utilise MinGW (gcc), installer l'ensemble de la distribution actuelle de

MinGW au moyen de l'installateur automatisé de MinGW depuis [www.mingw.org](http://www.mingw.org) [<http://www.mingw.org>], par exemple dans `c:/mingw`. Ceci installera gcc, g++, GNU binutils, les run-time de MinGW et l'API win32. Installer ensuite la version actuelle de MSys.

Sous Windows on peut utiliser Microsoft Visual C++ (sauf pour CsoundAC). L'Express Edition libre, depuis <http://www.microsoft.com/express/vc/> fonctionne très bien. Il vous faudra une copie du fichier d'en-tête de Windows `direct.h`, par exemple depuis <http://www.softagalleria.net/direct.php>. On peut aussi avoir besoin de la bibliothèque `bufferoverflowu.lib` de Microsoft à déposer dans le répertoire `lib` de Visual C++. Ouvrir ensuite un shell pour compiler Csound (habituellement appelé Visual Studio Command Prompt `command`, depuis le menu du programme Visual C++).

Les configurations optionnelles pour Windows comprennent :

- La bibliothèque multimedia de Windows pour l'audio en temps réel et le MIDI. Ce module sera construit automatiquement si les en-têtes sont trouvés.
- Les en-têtes VST de Steinberg pour les opcodes de l'hôte VST.

## Linux

Les configurations optionnelles pour Linux comprennent :

- ALSA ([www.alsa-project.org](http://www.alsa-project.org) [<http://www.alsa-project.org>]) et JACK ([www.jackaudio.org/](http://www.jackaudio.org/) [<http://www.jackaudio.org/>]) en plus de PortAudio, pour l'audio en temps-réel. Les distributions de linux fournissent habituellement les paquetages de développement pour ces systèmes dans leurs entrepôts.
- Les en-têtes LADSPA et DSSI pour les opcodes de l'hôte DSSI.

## Mac OS X

Les configurations optionnelles pour Mac OS X comprennent :

- CoreAudio (système audio natif d'OSX) et JACK, en plus de PortAudio, pour l'audio en temps-réel.
- Les en-têtes LADSPA et DSSI pour les opcodes de l'hôte DSSI.

## Construire Csound 5 avec SCons

Lorsque vous avez tous les paquetages requis et leur sources (ou les paquetages `-dev`) pour besoins particuliers sur votre plate-forme, exécutez `"scons -h"` pour découvrir les options de configuration.

La construction est considérablement facilitée si les bibliothèques et les en-têtes téléchargés sont installés dans leurs répertoires par défaut. Si l'on veut modifier la construction par défaut, en particulier pour prendre en compte les options non-standard des dépendances de tierces parties pour lesquelles il faut trouver les en-têtes et les bibliothèques :

- Sous Windows, si l'on utilise Microsoft Visual C++, modifier `custom-msvc.py`

- Sous Windows, si l'on utilise MinGW/MSys, modifier custom-mingw.py
- Sous Linux et Mac OSX éditer custom.py

Si vous modifiez ce fichier, marquez-le en lecture seule (c.à.d. protégez-le) afin que le CVS ne l'écrase pas lors d'une prochaine mise à jour des fichiers sources. Evitez de modifier le fichier SConstruct.

Exécutez scons avec les variables pour les options que vous désirez. Par exemple :

```
scons buildOSC=1 buildCsound5GUI=1 buildPythonOpcodes=1 useOSC=1 buildLoris=0
```



## Note

Il est important de positionner la variable d'environnement `OPCODEDIR` sur le répertoire dans lequel les bibliothèques de plugin se trouvent ; dans le cas d'une construction en double précision, il faut plutôt positionner `OPCODEDIR64`. Les installateurs s'occupent habituellement de ceci, mais Csound doit pouvoir trouver ses bibliothèques de plugin lorsqu'on le construit à partir des sources.

# Options de construction

**Tableau 4. Options de construction de SCons**

Variable d'ajustement	Effet si positionnée à 1
buildCsoundVST	Construire CsoundVST. Nécessite CsoundAC, FLTK, boost, Python, SWIG.
buildCsoundAC	Construire CsoundAC. Nécessite FLTK, boost, Python, SWIG.
buildCsound5GUI	Construire le frontal graphique FLTK. Nécessite FLTK 1.1.7 ou ultérieur.
buildCSEditor	Construire l'éditeur de texte avec coloration syntaxique de Csound. Nécessite les en-têtes et les bibliothèques de FLTK.
buildDSSI	Construire les opcodes de l'hôte DSSI/LADSPA.
buildImageOpcodes	Construire les opcodes d'image. 1 par défaut. Mettre à 0 pour désactiver.
buildInterfaces	Construire la bibliothèque d'interface pour Python, JAVA, Lua, C++ et d'autres langages.
buildJavaWrapper	Construire la sur-couche Java pour la bibliothèque d'interface.
buildLoris	Construire les opcodes et l'extension Python de Loris.
buildNewParser	Activer le nouveau parser. Nécessite Flex/Bison.
buildOSXGUI	Construire le frontal graphique de base. Seulement sous OSX.
buildPDClass	Construire la classe PD csoundapi~. Nécessite m_pd.h à l'endroit standard.
buildPythonOpcodes	Construire les opcodes Python
buildRelease	Construire en mode release. Positionne noDebug.

Variable d'ajustement	Effet si positionnée à 1
buildSDFT	Construire le code SDFT. 1 par défaut. Mettre à 0 pour désactiver.
buildTclcsound	Construire le frontal Tclcsound (cstclsh, cswish et le module dynamique tclcsound). Nécessite les en-têtes et les bibliothèques Tcl/Tk.
buildUtilities	Construire des exécutables autonomes pour les utilitaires que l'on peut aussi appeler avec -U.
buildVirtual	Construire le clavier virtuel MIDI. Nécessite les en-têtes et les bibliothèques de FLTK 1.1.7 ou ultérieur.
buildvst4cs	Construire les plugins vst4cs. Nécessite les en-têtes VST de Steinberg.
buildWinsound	Construire le frontal Winsound. Nécessite les en-têtes et les bibliothèques FLTK.
dynamicCsoundLibrary	Construire une bibliothèque Csound dynamique au lieu de libcsound.a.
gcc3opt	Autoriser les optimisations de gcc 3.3.x ou ultérieur pour l'architecture CPU spécifiée (par exemple pentium3) ; positionne noDebug.
gcc4opt	Autoriser les optimisations de gcc 4.0 ou ultérieur pour l'architecture CPU spécifiée (par exemple pentium3) ; positionne noDebug.
generateTags	Générer des TAGS.
generatePdf	Générer la documentation PDF.
install	Autoriser les cibles d'installation.
Lib64	Construire pour lib64 plutôt que pour lib.
noDebug	Construire sans information de débogage.
noFLTKThreads	Ne pas utiliser de thread séparé pour les contrôles graphiques de FLTK.
useAltiVec	Sous OSX, utiliser les options d'optimisation du gcc AltiVec.
useALSA	ALSA pour les entrées et les sorties audio en temps réel et MIDI.
useCoreAudio	Utiliser CoreAudio pour les entrées et les sorties audio en temps réel.
useDouble	Utiliser des nombres réels en double précision pour les échantillons audio.
useFLTK	Utiliser FLTK pour les graphiques et les opcodes de contrôle graphique.
useGettext	Utiliser le schéma de localisation de GNU
useGprof	Construire avec des informations de profilage (-pg).
usePortAudio	utiliser PortAudio pour les entrées et les sorties audio en temps réel.
usePortMIDI	Construire le plugin PortMidi pour les entrées et les sorties MIDI en temps réel.
useJack	A utiliser si vous avez compilé PortAudio pour utiliser Jack ; construit également le plugin Jack.

Variable d'ajustement	Effet si positionnée à 1
useLrint	Utiliser lrint() and lrintf() pour la conversion des nombres réels en entiers.
useOSC	Pour le support d'OSC.
useUDP	Pour le support d'UDP. 1 par défaut. Mettre à 0 pour désactiver.
withICL	Construire avec le compilateur C++ d'Intel (nécessite également Microsoft Visual C++). Fixer à 0 pour MinGW. Seulement sous Windows.
withMSVC	Construire avec Microsoft Visual C++, ou fixer à 0 pour construire avec MinGW. Seulement sous Windows.
Word64	Construire pour des machines 64 bit.
pythonVersion	Fixer à la version de Python que l'on veut utiliser.

---

# Liens Csound

La "page d'accueil" de Csound est maintenue par Richard Boulanger à <http://www.csounds.com>.

Le code source de Csound est maintenu par John ffitch et d'autres à <http://www.sourceforge.net/projects/csound>. Les versions les plus récentes et les paquetages précompilés pour la plupart des plates-formes peuvent être téléchargés ici [[http://sourceforge.net/project/showfiles.php?group\\_id=81968](http://sourceforge.net/project/showfiles.php?group_id=81968)].

Il existe une liste de diffusion Csound pour discuter de Csound. Elle est animée par John ffitch de Bath University, UK. Pour vous inscrire sur cette liste de diffusion envoyez un message vide à : [csound-subscribe@lists.bath.ac.uk](mailto:csound-subscribe@lists.bath.ac.uk) [<mailto:csound-subscribe@lists.bath.ac.uk>]. Vous pouvez aussi souscrire à la version condensée (1 message par jour) en envoyant un message vide à : [csound-digest-subscribe@lists.bath.ac.uk](mailto:csound-digest-subscribe@lists.bath.ac.uk) [<mailto:csound-digest-subscribe@lists.bath.ac.uk>]. Les messages envoyés à [csound@lists.bath.ac.uk](mailto:csound@lists.bath.ac.uk) [<mailto:csound@lists.bath.ac.uk>] sont distribués à tous les membres de la liste. On peut parcourir les archives de la liste de diffusion de Csound ici [[http://agentcities.cs.bath.ac.uk/%7ebwillkie/list\\_arch.php](http://agentcities.cs.bath.ac.uk/%7ebwillkie/list_arch.php)].

De même, la liste de diffusion `Csound-devel` existe pour discuter du développement de Csound. Pour plus d'information sur cette liste, aller à <http://lists.sourceforge.net/lists/listinfo/csound-devel>. Les messages envoyés à [csound-devel@lists.sourceforge.net](mailto:csound-devel@lists.sourceforge.net) [<mailto:csound-devel@lists.sourceforge.net>] vont à tous les membres de la liste.



---

## **Partie II. Vue d'Ensemble des Opcodes**

---

# Table des matières

Générateurs de Signal .....	95
Synthèse/Resynthèse Additive .....	95
Oscillateurs Elémentaires .....	95
Oscillateurs à Spectre Dynamique .....	95
Synthèse FM .....	96
Synthèse Granulaire .....	96
Synthèse Hyper Vectorielle .....	97
Générateurs Linéaires et Exponentiels .....	97
Générateurs d'Enveloppe .....	98
Modèles et Emulations .....	98
Phaseurs .....	99
Générateurs de Nombres Aléatoires (de Bruit) .....	100
Reproduction de Sons Echantillonnés .....	101
Soundfonts .....	101
Synthèse par Balayage .....	103
Accès aux Tables .....	104
Synthèse par Terrain d'Ondes .....	105
Modèles Physiques par Guide d'Onde .....	105
Entrée et Sortie de Signal .....	106
Entrées et Sorties Fichier .....	106
Entrée de Signal .....	106
Sortie de Signal .....	106
Bus Logiciel .....	107
Impression et Affichage .....	107
Requêtes sur les Fichiers Sons .....	107
Modificateurs de Signal .....	109
Modificateurs d'Amplitude et Traitement des Dynamiques .....	109
Convolution et Morphing .....	109
Retard .....	109
Panning et Spatialisation .....	110
Réverbération .....	112
Opérateurs du Niveau Echantillon .....	112
Limiteurs de Signal .....	113
Effets Spéciaux .....	113
Filtres Standard .....	113
Filtres Spécialisés .....	115
Guides d'Onde .....	115
Distorsion Non-Linéaire et Distorsion de Phase .....	115
Contrôle d'Instrument .....	117
Contrôle d'Horloge .....	117
Valeurs Conditionnelles .....	117
Instructions de Contrôle de Durée .....	117
Widgets FLTK et contrôleurs GUI .....	117
Conteneurs FLTK .....	120
Valuateurs FLTK .....	120
Autres Widgets FLTK .....	121
Modifier l'Apparence des Widgets FLTK .....	122
Opcodes Généraux relatifs aux Widgets FLTK .....	122
Appel d'Instrument .....	123
Contrôle Séquentiel d'un Programme .....	123
Contrôle de l'Exécution en Temps Réel .....	124
Initialisation et Réinitialisation .....	124
Détection et Contrôle .....	125

Piles .....	126
Contrôle de sous-instrument .....	126
Lecture du Temps .....	127
Contrôle des Tables de Fonction .....	128
Requêtes sur une Table .....	128
Opérations de Lecture/Ecriture de Table .....	128
Lecture de Table avec Sélection Dynamique .....	129
Opérations Mathématiques .....	130
Conversion d'Amplitude .....	130
Opérations Arithmétiques et Logiques .....	130
Comparateurs et Accumulateurs .....	130
Fonctions Mathématiques .....	131
Opcodes Equivalents à des Fonctions .....	131
Fonctions aléatoires .....	132
Fonctions Trigonométriques .....	132
Opcodes d'Algèbre Linéaire .....	133
Conversion des Hauteurs .....	143
Fonctions .....	143
Opcodes de Hauteurs .....	143
Support MIDI en Temps-Réel .....	144
Clavier Virtuel MIDI .....	145
Entrée MIDI .....	148
Sortie de Message MIDI .....	148
Entrée et Sortie Génériques .....	149
Convertisseurs .....	149
Extension d'Evènements .....	149
Sortie de Note-on/Note-off .....	149
Opcodes pour l'Interopérabilité MIDI/Partition .....	150
Messages System Realtime .....	151
Banques de Réglettes .....	151
Traitement Spectral .....	153
Resynthèse par Transformée de Fourier à Court-Terme (STFT) .....	153
Resynthèse par Codage Prédicatif Linéaire (LPC) .....	154
Traitement Spectral Non-standard .....	154
Outils pour le Traitement Spectral en Temps Réel (opcodes pvs) .....	154
Traitement Spectral avec ATS .....	156
Opcodes Loris .....	156
Chaînes de Caractères .....	161
Opcodes de Manipulation de Chaîne .....	162
Opcodes de Conversion de Chaîne .....	162
Opcodes Vectoriels .....	164
Opérateurs de Tableaux de Vecteurs .....	164
Opérations Entre un Signal Vectoriel et un Signal Scalaire .....	164
Opérations Entre deux Signaux Vectoriels .....	165
Générateurs Vectoriels d'Enveloppe .....	165
Limitation et Enroulement des Signaux Vectoriels de Contrôle .....	166
Chemins de Retard Vectoriel au Taux de Contrôle .....	166
Générateurs de Signal Aléatoire Vectoriel .....	166
Système de Patch Zak .....	168
Accueil de Plugin .....	169
DSSI et LADSPA pour Csound .....	169
VST pour Csound .....	169
OSC et Réseau .....	171
OSC .....	171
Réseau .....	171
Opcodes pour le Traitement à Distance .....	171
Opcodes Mixer .....	172
Opcodes de Graphe de Fluence .....	173

Opcodes Python .....	176
Introduction .....	176
Syntaxe de l'Orchestre .....	176
Opcodes pour le traitement d'image .....	178
Opcodes divers .....	179

---

# Générateurs de Signal

## Synthèse/Resynthèse Additive

Les opcodes pour la synthèse et la resynthèse additives sont :

- *adsyn*
- *adsynt*
- *adsynt2*
- *hsboscil*

Voir la section *Traitement Spectral* pour plus d'information et des opcodes de synthèse/re-synthèse additive supplémentaires.

## Oscillateurs Élémentaires

Les opcodes des oscillateurs élémentaires sont : (noter que les opcodes qui se terminent par un 'i' implémentent l'interpolation linéaire et que ceux qui se terminent par un '3' implémentent l'interpolation cubique)

- Banques d'Oscillateurs : *oscbnk*
- Oscillateurs simples à table : *oscil*, *oscil3* et *oscili*.
- Oscillateur sinusoïdal simple, rapide : *oscils*
- Oscillateurs de précision : *poscil* et *poscil3*.
- Oscillateurs plus flexibles : *oscilikt*, *osciliktp*, *oscilikts* et *osciln* (aussi appelé *oscilx*).

On peut aussi construire des oscillateurs à partir des opcodes de lecture de table. Voir la section *Opérations de Lecture/Ecriture de Table*.

## LFOs

- *lfo*
- *vibr*
- *vibrato*

Voir la section *Accès aux Tables* pour d'autres opcodes de lecture de table que l'on peut utiliser comme oscillateurs. Voir aussi la section *Oscillateurs à Spectre Dynamique*.

## Oscillateurs à Spectre Dynamique

Les opcodes qui génère des spectres dynamiques sont :

- Spectres harmoniques : *buzz* et *gbuzz*
- Générateur d'impulsions : *mpulse*
- Oscillateurs à bande limitée (d'après des modèles analogiques) : *vco* et *vco2*

On peut utiliser les opcodes suivants pour générer des formes d'onde à bande limitée pour une utilisation avec *vco2* et d'autres oscillateurs :

- *vco2init*
- *vco2ft*
- *vco2ift*

## Synthèse FM

Les opcodes de synthèse FM sont :

- *foscil*
- *foscili*
- *crossfm*, *crossfmi*, *crosspm*, *crosspmi*, *crossfmpm* et *crossfmpmi*.

## Modèles d'instrument FM

- *fmb3*
- *fmbell*
- *fmmetal*
- *fmpercfl*
- *fmrhode*
- *fmvoice*
- *fmwurlie*

## Synthèse Granulaire

Les opcodes de synthèse granulaire sont :

- *diskgrain*

- *fof*
- *fof2*
- *fog*
- *grain*
- *grain2*
- *grain3*
- *granule*
- *partikkel*
- *partikkelsync*
- *sndwarp*
- *sndwarpst*
- *syncgrain*
- *syncloop*
- *vosim*

## Synthèse Hyper Vectorielle

- *vphaseseg*
- *hvs1*
- *hvs2*
- *hvs3*

## Générateurs Linéaires et Exponentiels

Les opcodes qui génèrent des courbes ou des segments linéaires ou exponentiels sont :

- *expon*
- *expcurve*
- *expseg*
- *expsega*

- *expsegr*
- *gainslider*
- *jspline*
- *line*
- *linseg*
- *linsegr*
- *logcurve*
- *loopseg*
- *loopsegp*
- *lpshold*
- *lpsholdp*
- *rspline*
- *scale*
- *transeg*

## Générateurs d'Enveloppe

Les générateurs d'enveloppe suivants sont disponibles :

- *adsr*
- *madsr*
- *mxadsr*
- *xadsr*
- *linen*
- *linenr*
- *envlpx*
- *envlpxr*

Consulter la section des *Générateurs Linéaires et Exponentiels* pour d'autres méthodes de création d'enveloppes.

## Modèles et Emulations

Les opcodes suivants réalisent la modélisation ou l'émulation des sons d'autres instruments (certains basés sur la boîte à outils STK par Perry Cook) :



- *bamboo*
- *barmodel*
- *cabasa*
- *crunch*
- *dripwater*
- *gogobel*
- *guiro*
- *mandol*
- *marimba*
- *moog*
- *sandpaper*
- *sekere*
- *shaker*
- *sleighbells*
- *stix*
- *tambourine*
- *vibes*
- *voice*

Autres modèles et émulations

- *lorenz*
- *planet*
- *prepiano*
- Générateur de Nombres Fractals (ensemble de Mandelbrot) : *mandel*
- *chuap*

## Phaseurs

Les opcodes qui génèrent une valeur de phase mobile :

- *phasor*
- *phasorbnk*

- *syncphasor*

Ces opcodes sont utiles en combinaison avec les opcodes d'*Accès aux Tables*.

## Générateurs de Nombres Aléatoires (de Bruit)

Les opcodes qui génèrent des nombres aléatoires sont :

- *betarnd*
- *bexprnd*
- *cauchy*
- *cuserrnd*
- *duserrnd*
- *exprand*
- *gauss*
- *linrand*
- *noise*
- *pcauchy*
- *pinkish*
- *poisson*
- *rand*
- *randh*
- *randi*
- *rnd31*
- *random*
- *randomh*
- *randomi*
- *trirand*
- *unirand*
- *urd*
- *weibull*
- *jitter*
- *jitter2*

- *trandom*

Voir *seed* qui fixe la valeur de la racine globale pour tous les générateurs de bruit de classe *x*, ainsi que d'autres opcodes qui utilisent un appel de fonction aléatoire comme *grain.rand*, *randh*, *randi*, *rnd(x)* et *birnd(x)* ne sont pas affectés par *seed*.

Voir aussi les fonctions qui génèrent des nombres aléatoires dans la section *Fonctions Aléatoires*.

## Reproduction de Sons Echantillonnés

Les opcodes qui implémentent la reproduction de sons échantillonnés (samples) et les boucles sont :

- *bbcutm*
- *bbcuts*
- *flooper*
- *flooper2*
- *loscil*
- *loscil3*
- *loscilx*
- *lphasor*
- *lposcil*
- *lposcil3*
- *lposcila*
- *lposcilsa*
- *lposcilsa2*
- *sndloop*
- *waveset*

Voir aussi la section *Entrée de Signal* pour d'autres types d'entrées sonores.

## Soundfonts

### Opcodes Fluid

La famille des opcodes fluid encapsule le lecteur SoundFont 2 de Peter Hannape, FluidSynth : *fluidEngine* pour instancier un moteur FluidSynth, *fluidSetInterpMethod* pour fixer la méthode d'interpolation d'un canal dans un moteur FluidSynth, *fluidLoad* pour charger des SoundFonts, *fluidProgramSelect* pour assigner des presets d'un SoundFont à un canal MIDI d'un moteur FluidSynth, *fluidNote* pour jouer une note sur un canal MIDI d'un moteur FluidSynth, *fluidCCi* pour envoyer un message de contrôleur au temps-i sur un canal MIDI d'un moteur FluidSynth, *fluidCCK* pour envoyer un message de contrôleur au taux k sur un canal MIDI d'un moteur FluidSynth. *fluidControl* pour jouer et contrôler les Soundfonts

chargés (en utilisant des messages MIDI 'bruts'), *fluidOut* pour recevoir de l'audio depuis un seul moteur FluidSynth, et *fluidAllOut* pour recevoir de l'audio depuis tous les moteurs FluidSynth.

- *fluidAllOut*
- *fluidCCi*
- *fluidCCk*
- *fluidControl*
- *fluidEngine*
- *fluidLoad*
- *fluidNote*
- *fluidOut*
- *fluidProgramSelect*
- *fluidSetInterpMethod*

## Anciens opcodes Soundfont

Ces opcodes peuvent aussi employer des soundfonts pour générer du son. L'utilisation des opcodes fluid (ci-dessus) est recommandées plutôt que celle de ces opcodes.

- *sfilist*
- *sfinstr*
- *sfinstr3*
- *sfinstr3m*
- *sfinstrm*
- *sfload*
- *sfpassign*
- *sfplay*
- *sfplay3*
- *sfplay3m*
- *sfplaym*
- *sflooper*
- *sfplist*
- *sfpreset*

## Synthèse par Balayage

La synthèse par balayage (scanned synthesis) est une variante des modèles physiques, dans laquelle un réseau de masses connectées par des ressorts est utilisé pour générer une forme d'onde dynamique. L'opcode *scanu* définit le réseau de masses/ressorts et le met en mouvement. L'opcode *scans* suit un chemin prédéfini (une trajectoire) à travers le réseau et donne en sortie la forme d'onde détectée. Plusieurs instances de *scans* peuvent suivre différents chemins à travers le même réseau.

Ce sont des algorithmes de modélisation mécanique hautement efficaces à la fois pour la synthèse et l'animation sonore via un traitement algorithmique. Il vaut mieux les utiliser en temps réel. Ainsi, la sortie est utile soit directement pour l'audio, soit comme valeurs de contrôleur pour d'autres paramètres.

L'implémentation dans Csound ajoute le support pour un chemin de balayage ou matrice. Essentiellement, ceci offre la possibilité de reconnecter les masses dans d'autres configurations, provoquant une propagation du signal assez différente. Elles ne doivent pas nécessairement être connectées à leurs voisines directes. La matrice a essentiellement l'effet de « modeler » la surface en une forme radicalement différente.

Pour produire les matrices, le format du tableau est direct. Par exemple, pour 4 masses nous avons la grille suivante qui décrit les connexions possibles :

	1	2	3	4
1				
2				
3				
4				

Chaque fois que deux masses sont connectées, le point qu'elles définissent vaut 1. Si deux masses ne sont pas connectées, le point qu'elles définissent vaut alors 0. Par exemple, une corde unidirectionnelle a les connexions suivantes : (1,2), (2,3), (3,4). Si elle est bidirectionnelle, elle a aussi (2,1), (3,2), (4,3). Pour la corde unidirectionnelle, la matrice est :

	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0

Le format de tableau ci-dessus pour la matrice de connexion n'est donné que par commodité conceptuelle. Les valeurs actuellement montrées dans le tableau sont obtenues par *scans* depuis un fichier ASCII en utilisant *GEN23*. Le fichier ASCII lui-même est créé à partir du tableau modèle ligne par ligne. Ainsi, le fichier ASCII pour le tableau de l'exemple montré ci-dessus devient :

```
0100001000010000
```

Cet exemple de matrice est très simple et très petit. En pratique, la plupart des instruments de synthèse par balayage utiliseront bien plus que quatre masses, et donc leurs matrices seront bien plus grandes et plus complexes. Voir l'exemple dans la documentation de *scans*.

Prière de noter que les tables d'onde dynamiques générées sont très instables. Certaines valeurs de masses, de centrage, et d'amortissement peuvent provoquer une « explosion » du système et l'apparition

des sons les plus intéressants sur vos haut-parleurs.

Le supplément de ce manuel contient un tutoriel sur la synthèse par balayage. Le tutoriel, des exemples, et d'autres informations sur la synthèse par balayage sont disponibles sur la page Scanned Synthesis à [csounds.com](http://www.csounds.com/scanned/) [http://www.csounds.com/scanned/].

La synthèse par balayage a été développée par Bill Verplank, Max Mathews et Rob Shaw à Interval Research entre 1998 et 2000.

Les opcodes qui implémentent la synthèse par balayage sont :

- *scanhammer*
- *scans*
- *scantable*
- *scanu*
- *xscanmap*
- *xscans*
- *xscansmap*
- *xscanu*

## Accès aux Tables

Les opcodes qui permettent l'accès aux tables sont :

- *oscill*
- *oscilli*
- *osciln*
- *oscilx*
- *table*
- *table3*
- *tablei*

Les opcodes se terminant par 'i' implémentent l'interpolation linéaire et les opcodes se terminant par '3' implémentent l'interpolation cubique.

Les opcodes suivants implémentent la lecture/écriture rapide dans une table sans en tester les limites :

- *tab*
- *tab\_i*
- *tabw*

- *tabw\_i*

Voir les sections *Requêtes de Table*, *Opérations de Lecture/Ecriture de Table* et *Lecture de Table avec Sélection Dynamique* pour d'autres opérations de table.



### Note

Bien que des tables avec une taille qui n'est pas une puissance de deux puissent être créées en utilisant une taille négative (voir *instruction de partition f*), certains opcodes ne les accepteront pas.

## Synthèse par Terrain d'Ondes

L'opcode qui utilise la synthèse par terrain d'ondes est : *wterrain*.

## Modèles Physiques par Guide d'Onde

Les opcodes qui implémentent les modèles physiques par guide d'onde sont :

- *pluck*
- *repluck*
- *wgbow*
- *wgbowedbar*
- *wgbrass*
- *wgclar*
- *wgflute*
- *wgpluck*
- *wgpluck2*
- *wguide1*
- *wguide2*

---

# Entrée et Sortie de Signal

## Entrées et Sorties Fichier

Les opcodes pour les entrées et sorties fichier sont :

- Ouverture/fermeture de fichier : *fiopen* et *ficlose*.
- Sortie fichier : *dumpk*, *dumpk2*, *dumpk3*, *dumpk4*, *fout*, *fouti*, *foutir* et *foutk*
- Entrée fichier : *readk*, *readk2*, *readk3*, *readk4*, *fin*, *fini* et *fink*
- Utilitaires à utiliser avec les opcodes *fout* : *clear*, *vincr*
- Impression dans un fichier : *fprints* et *fprintks*

## Entrée de Signal

Les opcodes qui reçoivent des signaux audio sont :

- Entrée synchrone : *in*, *in32*, *inch*, *inh*, *ino*, *inq*, *inrg*, *ins* et *inx*
- Flux de fichier : *diskin*, *diskin2* et *soundin*
- Canal d'entrée défini par l'utilisateur : *invalue*
- Flux d'entrée : *soundin*
- Entrée directe dans *zak* : *inz*

Voir la section *Bus Logiciel* pour les entrées et les sorties au moyen de l'API.

*mp3in* permet la lecture des fichiers mp3, qui n'est pas supportée par les méthodes de lecture usuelles dans Csound.

## Sortie de Signal

Les opcodes qui écrivent des signaux audio sont :

- Sortie synchrone : *out*, *out32*, *outc*, *outch*, *outh*, *outo*, *outrg*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2* et *outx*
- Flux de sortie : *soundout* et *soundouts*
- Canal de sortie défini par l'utilisateur : *outvalue*
- Sortie directe depuis *zak* : *outz*

L'opcode *monitor* peut être utilisé pour surveiller la sortie complète de csound (trame de sor-



tie spout).

Voir la section *Bus Logiciel* pour les entrées et les sorties au moyen de l'API.

## Bus Logiciel

Csound implémente un bus logiciel pour le routage interne ou le routage vers des logiciels externes en appelant l'API de Csound.

Les opcodes pour utiliser le bus logiciel sont :

- *chn\_k*
- *chn\_a*
- *chn\_S*
- *chnclear*
- *chnexport*
- *chnmix*
- *chnparams*

## Impression et Affichage

Les opcodes pour imprimer et afficher des valeurs sont :

- *dispfft*
- *display*
- *flashtxt*
- *print*
- *printf*
- *printf\_i*
- *printk*
- *printk2*
- *printks*
- *prints*

## Requêtes sur les Fichiers Sons

Les opcodes qui demandent de l'information sur les fichiers sont :

- *filelen*
- *filenchnls*
- *filepeak*
- *filesr*

---

# Modificateurs de Signal

## Modificateurs d'Amplitude et Traitement des Dynamiques

Les opcodes qui modifient l'amplitude sont :

- *balance*
- *compress*
- *clip*
- *dam*
- *gain*

L'opcode *Odbfs* facilite la manipulation d'amplitude en supprimant la nécessité d'utiliser des valeurs d'échantillon explicites.

## Convolution et Morphing

Les opcodes qui font la convolution et le morphing de signaux sont :

- *convolve* aussi nommé *convle*
- *cross2*
- *dconv*
- *ftconv*
- *ftmorf*
- *pconvolve*

## Retard

### Retards fixes

- *delay*
- *delay1*
- *delayk*

## Lignes à retard

- *delayr*
- *delayw*
- *deltap*
- *deltap3*
- *deltapi*
- *deltapn*
- *deltapx*
- *deltapxw*

## Retards variables

- *vdelay*
- *vdelay3*
- *vdelayx*
- *vdelayxs*
- *vdelayxq*
- *vdelayxw*
- *vdelayxwq*
- *vdelayxws*

## Retards multiples

- *multitap*

## Panning et Spatialisation

### Spatialisation d'Amplitude

- *locsend*
- *locsig*
- *pan*

- *pan2*
- *space*
- *spdist*
- *spsend*

## Spatialisation 3D avec simulation d'acoustique des salles

- *spat3d*
- *spat3di*
- *spat3dt*

## Panning d'Amplitude à Base Vectorielle

- *vbap16*
- *vbap16move*
- *vbap4*
- *vbap4move*
- *vbap8*
- *vbap8move*
- *vbaplsinit*
- *vbapz*
- *vbapzmove*

## Spatialisation Binaurale

- *hrtfer*
- *hrtfmove*
- *hrtfmove2*
- *hrtfstat*

## Ambisonics

- *bformdec*
- *bformenc*

## Réverbération

Les opcodes qu'on peut utiliser pour la réverbération sont :

- *alpass*
- *babo*
- *comb*
- *freeverb*
- *nestedap*
- *nreverb* (aussi appelé *reverb2*)
- *reverb*
- *reverbse*
- *valpass*
- *vcomb*

## Opérateurs du Niveau Echantillon

Les opérateurs que l'on peut utiliser pour modifier les signaux sont :

- *a(k)*
- *denorm*
- *diff*
- *downsamp*
- *fold*
- *i(k)*
- *integ*
- *interp*
- *i(k)*
- *ntrpol*
- *samphold*

- *upsamp*
- *vaget*
- *vaset*

## Limiteurs de Signal

Les opcodes que l'on peut utiliser pour limiter des signaux sont :

- *limit*
- *mirror*
- *wrap*

## Effets Spéciaux

Les opcodes qui génèrent des effets spéciaux sont :

- *distort*
- *distort1*
- *flanger*
- *harmon*
- *phaser1*
- *phaser2*

## Filtres Standard

### Filtres passe-bas à résonance

- *areson*
- *lowpass2*
- *lowres*
- *lowresx*
- *lpf18*
- *moogvcf*
- *moogladder*

- *reson*
- *resonr*
- *resonx*
- *resony*
- *resonz*
- *rezzy*
- *statevar*
- *svfilter*
- *tbvcf*
- *vlowres*
- *bqrez*

## Filtres standard

- Filtres passe-haut : *atone*, *atonex*
- Filtres passe-bas : *tone*, *tonex*
- Filtres biquadratiques : *biquad* et *biquada*.
- Filtres de Butterworth : *butterbp*, *butterbr*, *butterhp*, *butterlp* (qui sont aussi appelés *butbp*, *butbr*, *buthp*, *butlp*)
- Filtres généraux : *clfilt*

## Filtres de signal de contrôle

- *aresonk*
- *atonek*
- *lineto*
- *port*
- *portk*
- *resonk*
- *resonxk*
- *tlineto*
- *tonek*



## Filtres Spécialisés

### Filtres passe-haut

- *dcblock*
- *dcblock2*

### Egaliseurs paramétriques

- *pareq*
- *rbjeq*
- *eqfil*

### Autres filtres

- *nlfilt*
- *filter2*
- *fofilter*
- *hilbert*
- *zfilter2*

## Guides d'Onde

Les opcodes qui utilisent des guides d'onde pour modifier un signal sont :

- *streson*
- *wguide1*
- *wguide2*

## Distorsion Non-Linéaire et Distorsion de Phase

Ces opcodes peuvent exécuter de façon dynamique une distorsion non-linéaire ou une distorsion de phase. Il diffèrent des méthodes traditionnelles de distorsion non-linéaire basées sur une table, en calculant directement la fonction de transfert avec un ou plusieurs paramètres variables pour modifier l'importance ou les résultats de la distorsion. La plupart de ces opcodes peuvent être utilisés sur un signal audio (pour la distorsion non-linéaire) ou sur un phaseur (pour la distorsion de phase) mais ils ont tendance à fonctionner au mieux pour une de ces applications.

Ces opcodes sont adaptés à la distorsion non-linéaire :

- *chebyshevpoly*
- *clip*
- *distort*
- *distort1*
- *polynomial*
- *powershape*

Ces opcodes sont adaptés à la distorsion de phase :

- *pdclip*
- *pdhalf*
- *pdhalfy*

---

# Contrôle d'Instrument

## Contrôle d'Horloge

Les opcodes pour démarrer et arrêter les horloges internes sont :

- *clockoff*
- *clockon*

Ces horloges comptent le temps CPU. On dispose de 32 horloges indépendantes. On peut utiliser l'opcode *readclock* pour lire les valeurs courantes d'une horloge. Voir *Lecture du Temps* pour d'autres opcodes de chronométrage.

## Valeurs Conditionnelles

Les opcodes pour les valeurs conditionnelles sont `==`, `>=`, `>`, `<`, `<=` et `!=`.

## Instructions de Contrôle de Durée

Les opcodes que l'on peut utiliser pour manipuler la durée d'une note sont :

- *ihold*
- *turnoff*
- *turnoff2*
- *turnon*

Pour d'autres contrôles d'instrument en temps réel voir *Contrôle de l'Exécution en Temps Réel* et *Appel d'Instrument*.

## Widgets FLTK et contrôleurs GUI

Les widgets permettent de dessiner une Interface Utilisateur Graphique (GUI) personnalisée pour contrôler un orchestre en temps réel. Ils sont dérivés de la bibliothèque libre FLTK (Fast Light ToolKit). Cette bibliothèque est une des plus rapides parmi les bibliothèques disponibles, supporte OpenGL et devrait être compatible avec différentes plates-formes (Windows, Linux, Unix et Mac OS). Le sous-ensemble de FLTK implémenté dans Csound fournit les types d'objets suivants :

Conteneurs

Les *Conteneurs FLTK* sont des widgets qui contiennent d'autres widgets tels que des panneaux, des fenêtres, etc. Csound fournit les objets conteneurs suivants :

- Panneaux
- Zones déroulantes

	<ul style="list-style-type: none"><li>• Paquets</li><li>• Onglets</li><li>• Groupes</li></ul>
Valuateurs	<p>Les objets les plus utiles sont appelés <i>Valuateurs FLTK</i>. Ces objets permettent à l'utilisateur de modifier les valeurs des paramètres de synthèse en temps réel. Csound fournit les objets valuateurs suivants :</p> <ul style="list-style-type: none"><li>• Réglettes</li><li>• Boutons rotatifs</li><li>• Molettes</li><li>• Champs texte</li><li>• Joysticks</li><li>• Compteurs</li></ul>
Autres widgets	<p>Il y a d'autres <i>widgets FLTK</i> qui ne sont ni des valuateurs ni des conteneurs :</p> <ul style="list-style-type: none"><li>• Boutons</li><li>• Bancs de boutons</li><li>• Etiquettes</li><li>• Détection Clavier et Souris</li></ul>

Il y a aussi d'autres opcodes utiles pour modifier l'apparence des *widgets* :

- Mettre à jour la valeur d'un widget.
- Choisir les couleurs principale et de sélection d'un widget.
- Choisir le type, la taille et la couleur de police des widgets.
- Redimensionner un widget.
- Cacher et Montrer un widget.

Il y a aussi ces *opcodes généraux* qui permettent les actions suivantes :

- Lancer le processus léger (thread) des widgets : *FLrun*
- Charger des instantanés contenant l'état de tous les valuateurs d'un orchestre : *FLgetsnap* et *FLloadsnap*.
- Sauvegarder des instantanés contenant l'état de tous les valuateurs d'un orchestre : *FLsavesnap* et *FLsetsnap*
- Fixer le groupe d'instantanés d'un valuateur déclaré : *FLsetSnapGroup*

Ci-dessous un exemple simple de code Csound pour créer une fenêtre. Noter que tous les opcodes sont de taux-init et ne doivent être appelés qu'une seule fois par session. La meilleure manière de les utiliser est de les placer dans la section d'en-tête de l'orchestre, avant tout instrument. Même s'il n'est pas interdit de les placer dans un instrument, cela peut conduire à des résultats imprévisibles si l'instrument est appelé plus d'une fois.

Chaque conteneur est fait d'un couple d'opcodes : le premier indique le début du bloc du conteneur et le deuxième indique la fin du bloc du conteneur. Certains blocs de conteneur peuvent être imbriqués mais il ne peuvent pas se chevaucher. Après avoir défini tous les conteneurs, il faut lancer un processus léger de widgets en utilisant l'opcode spécial *FLrun* qui ne prend pas d'argument.

```
<CsoundSynthesizer>
<CsOptions>
; Sélectionner les options audio/midi ici, en fonction de la plate-forme
; Sortie audio   Entrée audio   Pas de messages
; -odac          -iadc          -d          ;; E/S audio en Temps Réel
; Pour une sortie différée ne garder que la ligne ci-dessous :
; -o linseg.wav -W ;; pour une sortie dans un fichier sur toute plate-forme
</CsOptions>
<CsInstruments>
;*****
sr=48000
kr=480
ksmps=100
nchnls=1

;*** Il est recommandé de placer presque tout le code GUI dans la
;*** section d'en-tête de l'orchestre

        FLpanel          "Panel1",450,550 ;***** début du conteneur
; placer ici quelques widgets
        FLpanelEnd       ;***** fin du conteneur

        FLrun            ;***** lance le thread FLTK, toujours requis !
instr 1
; placer ici du code de synthèse
endin
;*****
</CsInstruments>
<CsScore>
f 0 3600 ; table bidon pour l'entrée en temps réel
e

</CsScore>
</CsoundSynthesizer>
```

Le code précédent crée simplement un panneau (une fenêtre vide car aucun widget n'est défini à l'intérieur du conteneur).

L'exemple suivant crée deux panneaux et insère une réglette dans chacun d'entre eux :

```
<CsoundSynthesizer>
<CsOptions>
; Sélectionner les options audio/midi ici, en fonction de la plate-forme
; Sortie audio   Entrée audio   Pas de messages
; -odac          -iadc          -d          ;; E/S audio en Temps Réel
; Pour une sortie différée ne garder que la ligne ci-dessous :
; -o linseg.wav -W ;; pour une sortie dans un fichier sur toute plate-forme
</CsOptions>
<CsInstruments>
;*****
sr=48000
kr=480
ksmps=100
nchnls=1

        FLpanel          "Panel1",450,550,100,100 ;***** début de conteneur
gkl, iha FLslider        "FLslider 1", 500, 1000, 0 ,1, -1, 300,15, 20,50
```

```

        FLpanelEnd      ;***** fin de conteneur

        FLpanel        "Panel2",450,550,100,100 ;***** début de conteneur
gk2,ihb FLslider        "FLslider 2", 100, 200, 0 ,1, -1, 300,15, 20,50
        FLpanelEnd      ;***** fin de conteneur

        FLrun          ;***** lance le thread FLTK, toujours requis !

instr 1
; les variables gk1 et gk2 qui contiennent les valeurs de sortie des valuateurs
; définis précédemment, peuvent être utilisées à l'intérieur des instruments
printk2 gk1
printk2 gk2 ; imprime les valeurs des valuateurs chaque fois qu'elles changent
endin
;*****
</CsInstruments>
<CsScore>
f 0 3600 ; table bidon pour l'entrée en temps réel
e

</CsScore>
</CsoundSynthesizer>

```

Tous les opcodes de widget sont des opcodes de taux-init, même si les valuateurs donnent en sortie des variables de taux-k. Ceci est dû au fait qu'un processus léger indépendant est exécuté sur la base d'un mécanisme de fonctions de rappel. Cela permet de consommer très peu de ressources système car on évite la scrutation. (A la différence des autres opcodes de contrôleurs basés sur le MIDI). On peut ainsi utiliser n'importe quel nombre de fenêtres et de valuateurs sans dégrader l'exécution en temps réel.

## Conteneurs FLTK

Les opcodes pour les conteneurs FLTK sont :

- *FLgroup*
- *FLgroupEnd*
- *FLpack*
- *FLpackEnd*
- *FLpanel*
- *FLpanelEnd*
- *FLscroll*
- *FLscrollEnd*
- *FLtabs*
- *FLtabsEnd*

## Valuateurs FLTK

Les opcodes pour les valuateurs FLTK sont :

- *FLcount*

- *FLjoy*
- *FLknob*
- *FLroller*
- *FLslider*
- *FLtext*

## Autres Widgets FLTK

Les opcodes des autres widgets FLTK sont :

- *FLbox*
- *FLbutBank*
- *FLbutton*
- *FLexecButton*
- *FLkeyIn*
- *FLhvsBox*
- *FLhvsBoxSetValue*
- *FLmouse*
- *FLprintk*
- *FLprintk2*
- *FLslidBnk*
- *FLslidBnk2*
- *FLslidBnkGetHandle*
- *FLslidBnkSet*
- *FLslidBnk2Set*
- *FLslidBnk2Setk*
- *FLvalue*
- *FLvkeybd*
- *FLvslidBnk*
- *FLvslidBnk2*
- *FLxyin*

## Modifier l'Apparence des Widgets FLTK

Les opcodes suivants modifient l'apparence des widgets FLTK :

- *FLcolor*
- *FLcolor2*
- *FLhide*
- *FLlabel*
- *FLsetAlign*
- *FLsetBox*
- *FLsetColor*
- *FLsetColor2*
- *FLsetFont*
- *FLsetPosition*
- *FLsetSize*
- *FLsetText*
- *FLsetTextColor*
- *FLsetTextSize*
- *FLsetTextType*
- *FLsetVal\_i*
- *FLsetVal*
- *FLshow*

## Opcodes Généraux relatifs aux Widgets FLTK

Les opcodes généraux relatifs aux widgets FLTK sont :

- *FLgetsnap*
- *FLloadsnap*
- *FLrun*
- *FLsavesnap*
- *FLsetsnap*
- *FLupdate*
- *FLsetSnapGroup*



## Appel d'Instrument

Les opcodes que l'on peut utiliser pour créer des événements de partition depuis un orchestre sont :

- *event*
- *event\_i*
- *scoreline\_i*
- *scoreline*
- *schedule*
- *schedwhen*
- *schedkwhen*
- *schedkwhennamed*

L'opcode *mute* peut être utilisé pour rendre silencieux/sonore un instrument pendant une exécution.

Les définitions d'instrument peuvent être supprimées au moyen de l'opcode *remove*.

## Contrôle Séquentiel d'un Programme

Les opcodes pour modifier l'ordre d'exécution des instructions de l'orchestre sont :

- *cgoto*
- *cigoto*
- *ckgoto*
- *cngoto*
- *elseif*
- *else*
- *endif*
- *goto*
- *if*
- *igoto*
- *kgoto*
- *tigoto*
- *timeout*

Les opcodes pour créer des structures de boucle sont :

- *loop\_ge*
- *loop\_gt*
- *loop\_le*
- *loop\_lt*



### Avertissement

Certains de ces opcodes fonctionnent au taux-i même s'ils contiennent des comparaisons aux taux-k ou -a. Voir la section *Réinitialisation*.

## Contrôle de l'Exécution en Temps Réel

Les opcodes qui surveillent et contrôlent l'exécution en temps réel sont :

- *active*
- *cpuprc*
- *maxalloc*
- *prealloc*
- *jacktransport*

Le processus csound en cours peut être terminé au moyen de *exitnow*.

## Initialisation et Réinitialisation

Les opcodes utilisés pour l'initialisation des variables sont :

- *init*
- *tival*
- *=*
- *passign*
- *pset*

Les opcodes qui peuvent générer une autre passe d'initialisation sont :

- *reinit*
- *rigoto*

- *rireturn*

L'opcode *p* peut être utilisé pour lire les valeurs des p-champs aux taux-i ou -k.

*nstrnum* retourne le numéro d'instrument d'un instrument nommé.

## Détection et Contrôle

### Widgets TCL/TK

- *button*
- *checkbox*
- *control*
- *setctrl*

### Détection clavier et souris

- *sensekey* (aussi appelé *sense*)
- *xyin*

### Suiveurs d'enveloppe

- *follow*
- *follow2*
- *peak*
- *rms*

### Estimation de Tempo et de Hauteur

- *ptrack*
- *pitch*
- *pitchamdf*
- *tempest*

## Tempo et Séquencement

- *tempo*
- *miditempo*
- *tempoval*
- *seqtime*
- *seqtime2*
- *trigger*
- *trigseq*
- *timedseq*
- *changed*

## Système

- *getcfg*

## Contrôle de la partition

- *rewindscore*
- *setscorepos*

## Piles

Csound implémente une pile globale qui peut être manipulée par les opcodes suivants :

- *stack*
- *pop*
- *push*
- *pop\_f*
- *push\_f*

## Contrôle de sous-instrument

Ces opcodes permettent la définition et l'utilisation d'un sous-instrument :

- *subinstr*
- *subinstrinit*

Voir aussi les sections *UDO* et *Macros d'Orchestre* pour des fonctionnalités similaires.

## Lecture du Temps

Les opcodes que l'on peut utiliser pour lire des valeurs temporelles sont :

- *readclock*
- *rtclock*
- *timeinstk*
- *timeinsts*
- *times*
- *timek*

On peut obtenir la date du système au moyen de :

- *date* - Retourne le nombre de secondes écoulées depuis le 1er janvier 1970.
- *dates* - Retourne sous format chaîne la date et le temps spécifiés.

On peut aussi mettre en place des compteurs au moyen de *clockoff* et de *clockon*.

---

# Contrôle des Tables de Fonction

Se reporter aux sections *Instruction de partition f*, *ftgen*, *ftgentmp*, *figenonce* et *Routines GEN* pour savoir comment créer des tables.

On peut supprimer des tables de la mémoire au moyen de l'opcode *ftfree*.

Les tables requièrent par défaut une taille qui est une puissance de deux. On peut cependant générer des tables de n'importe quelle taille en spécifiant celle-ci comme un nombre négatif (voir l'*instruction de partition f*).



## Note

Certains opcodes n'acceptent pas des tables dont la taille n'est pas une puissance de deux, car ceci peut être une nécessité pour le traitement interne.

Pour savoir comment accéder aux tables, consulter la section *Accès aux Tables*.

Les tables à utiliser avec l'opcode *loscilx* peuvent être chargées au moyen de *sndload*.

## Requêtes sur une Table

Les opcodes qui permettent d'obtenir des informations sur une table sont :

- Pour les tables chargées avec le contenu d'un fichier son (au moyen de *GEN01*) : *fchnls*, *flen*, *filptim* et *ftsr*
- Pour n'importe quelle table : *nsamp*, *flen*, *tableng*

L'opcode *tabsum* calcule la somme des valeurs dans une table.

## Opérations de Lecture/Ecriture de Table

Les opcodes pour la lecture et l'écriture dans une table sont :

- *ftloadk*
- *ftload*
- *ftsavek*
- *ftsave*
- *tablecopy*
- *tablegpw*
- *tableicopy*
- *tableigpw*
- *tableimix*

- *tableiw*
- *tablemix*
- *tablera*
- *tablew*
- *tablewa*
- *tablewkt*
- *tabmorph*
- *tabmorpha*
- *tabmorphak*
- *tabmorphi*
- *tabrec*
- *tabplay*
- *ftmorf*

Les valeurs d'une table peuvent être lues depuis une expression grâce à la famille d'opcodes *tb*.

Plusieurs oscillateurs sont en fait des lecteurs de table spécialisés. Voir la section *Oscillateurs Elémentaires*.

## Lecture de Table avec Sélection Dynamique

Les opcodes qui permettent de sélectionner des tables dynamiquement (au taux-k) sont :

- *tableikt*
- *tablekt*
- *tablexkt*

---

# Opérations Mathématiques

## Conversion d'Amplitude

Les opcodes pour opérer des conversions entre différentes mesures d'amplitude sont :

- *ampdb*
- *ampdbfs*
- *db*
- *dbamp*
- *dbfsamp*

Utiliser *rms* pour trouver la valeur de la moyenne quadratique d'un signal. Voir aussi *0dbfs* pour un autre moyen de gérer les amplitudes dans *csound*.

## Opérations Arithmétiques et Logiques

Les opcodes qui effectuent les opérations arithmétiques et logiques sont : -, +, &&, //, \*, /, ^ et %.

Voir aussi la section *Valeurs Conditionnelles* et la famille des opcodes *if* pour l'utilisation des opérateurs logiques.

## Comparateurs et Accumulateurs

Les opcodes suivants effectuent la comparaison entre des signaux de taux-a ou de taux-k, trouvent les maxima ou les minima, ou accumulent les résultats de plusieurs calculs ou comparaisons :

- *max*
- *max\_k*
- *maxabs*
- *maxabsaccum*
- *maxaccum*
- *min*
- *minabs*
- *minabsaccum*
- *minaccum*
- *vincr*



- *clear*

## Fonctions Mathématiques

Les opcodes qui réalisent les fonctions mathématiques sont :

- *abs*
- *ceil*
- *exp*
- *floor*
- *frac*
- *int*
- *log*
- *log10*
- *logbtwo*
- *pow*
- *powershape*
- *powoftwo*
- *round*
- *sqrt*

## Opcodes Equivalents à des Fonctions

Les opcodes suivants sont équivalents à des fonctions mathématiques :

- *chebyshevpoly*
- *divz*
- *mac*
- *maca*
- *polynomial*
- *pow*
- *product*
- *sum*

- *taninv2*

## Fonctions aléatoires

Les opcodes qui effectuent des fonctions aléatoires sont :

- *birnd*
- *rnd*

Voir la section *Générateurs de Nombres Aléatoires (Bruit)* pour les opcodes qui génèrent des signaux aléatoires.

## Fonctions Trigonométriques

Les opcodes qui effectuent les fonctions trigonométriques sont :

- *cos*, *cosh* et *cosinv*
- *sin*, *sinh* et *sininv*
- *tan*, *tanh*, *taninv* et *taninv2*.

# Opcodes d'Algèbre Linéaire

Opcodes d'Algèbre Linéaire — Arithmétique scalaire, vectorielle et matricielle sur des valeurs réelles et complexes.

## Description

Ces opcodes implémentent plusieurs opérations d'algèbre linéaire, depuis l'arithmétique scalaire, vectorielle et matricielle jusqu'aux décompositions en valeurs propres basées sur la décomposition QR. Les opcodes sont conçus pour le traitement numérique du signal, et bien sûr pour d'autres opérations mathématiques, dans le langage d'orchestre de Csound.

L'implémentation numérique utilise la bibliothèque `gmm++` de [home.gna.org/getfem/gmm\\_intro](http://home.gna.org/getfem/gmm_intro) [[http://home.gna.org/getfem/gmm\\_intro](http://home.gna.org/getfem/gmm_intro)].



### Avertissement

Pour les applications avec des variables f-sig, l'arithmétique sur les tableaux ne peut être exécutée que si le f-sig est "actuel", car le taux-f est une fraction du taux-k ; ce caractère actuel peut être déterminé avec l'opcode `la_k_current_f`.

Pour les applications que utilisent des affectations entre vecteurs réels et variables de taux-a, l'arithmétique sur les tableaux ne peut être exécutée que si les vecteurs sont "actuels", car la taille du vecteur peut être un multiple entier de ksmps ; ce caractère actuel peut être déterminé au moyen de l'opcode `la_k_current_vr`.

**Tableau 5. Types de Données de l'Algèbre Linéaire**

Type Mathématique	Code	Type(s) de Csound Correspondant(s)
scalaire réel	r	variable de taux-i ou de taux-k
scalaire complexe	c	paire de variables de taux-i ou de taux-k, par exemple "kr, ki"
vecteur réel	vr	variable de taux-i contenant l'adresse d'un tableau
vecteur réel	a	variable de taux-a
vecteur réel	t	numéro d'une table de fonction
vecteur complexe	vc	variable de taux-i contenant l'adresse d'un tableau
vecteur complexe	f	variable fsig
matrice réelle	mr	variable de taux-i contenant l'adresse d'un tableau
matrice complexe	mc	variable de taux-i contenant l'adresse d'un tableau

Tous les tableaux sont indexés à partir de 0 ; le premier indice parcourt les lignes pour donner les colonnes, le deuxième indice parcourt les colonnes pour donner les éléments.



# Examen de Tableau

```
irows      la_i_size_vr      ivr
```

Retourne le nombre de lignes du vecteur réel *ivr*.

```
irows      la_i_size_vc      ivc
```

Retourne le nombre de lignes du vecteur complexe *ivc*.

```
irows, icolumns      la_i_size_mr      imr
```

Retourne le nombre de lignes et de colonnes de la matrice réelle *imr*.

```
irows, icolumns      la_i_size_mc      imc
```

Retourne le nombre de lignes et de colonnes de la matrice complexe *imc*.

kfiscurrent	la_k_current_f	fsig
-------------	----------------	------

Retourne 1 si le fsig est actuel, c'est-à-dire si la valeur du fsig changera lors de la prochaine période-k.

kvriscurrent                      la\_k\_current\_vr                      ivr

Retourne 1 si le vecteur réel est actuel, c'est-à-dire, si la trame d'échantillon actuelle de Csound se trouve à l'indice 0 du vecteur.

```
la_i_print_vr      ivr
```

Affiche la valeur du vecteur réel *ivr*.

```
la_i_print_vc      ivc
```

Affiche la valeur du vecteur complexe *ivc*.

```
la_i_print_mr      imr
```

Affiche la valeur de la matrice réelle  $imr$ .

```
la_i_print_mc      imc
```

Affiche la valeur de la matrice complexe *imc*.

## Affectation et Conversion de Tableau

```
ivr                                la_i_assign_vr                                ivr
```

Affecte la valeur du vecteur réel à droite au vecteur réel à gauche, au taux- $i$ .

la k assign vr

Affecte la valeur du vecteur réel à droite au vecteur réel à gauche, au taux-k.

ivc                                      la\_i\_assign\_vc                                      ivc

ivc                                      la\_k\_assign\_vc                                      ivr

imr	la_i_assign_mr	imr
imr	la_k_assign_mr	imr
imc	la_i_assign_mc	imc
imc	la_k_assign_mc	imr



## Avertissement

Les affectations vers des vecteurs à partir de tables ou de fsigs peuvent reformater les vecteurs.

Les affectations vers des vecteurs à partir de variables de taux-a, ou vers des variables de taux-a à partir de vecteurs, seront exécutées de manière incrémentielle, un bloc de ksmps éléments par période-k. C'est pourquoi l'arithmétique vectorielle sur ces vecteurs ne peut être pratiquée que si ceux-ci sont actuels, selon la détermination par l'opcode *la\_k\_current\_vr*.

ivr	la_k_assign_a	asig
ivr	la_i_assign_t	itablenumber
ivr	la_k_assign_t	itablenumber
ivc	la_k_assign_f	fsig
asig	la_k_a_assign	ivr
itablenum	la_i_t_assign	ivr
itablenum	la_k_t_assign	ivr
fsig	la_k_f_assign	ivc

## Remplissage des Tableaux par des Éléments Aléatoires

ivr	la_i_random_vr	[ifill_fraction]
ivr	la_k_random_vr	[kfill_fraction]
ivc	la_i_random_vc	[ifill_fraction]
ivc	la_k_random_vc	[kfill_fraction]
imr	la_i_random_mr	[ifill_fraction]
imr	la_k_random_mr	[kfill_fraction]
imc	la_i_random_mc	[ifill_fraction]
imc	la_k_random_mc	[kfill_fraction]

## Accès aux Éléments d'un Tableau

ivr	la_i_vr_set	irow, ivalue
kvr	la_k_vr_set	krow, kvalue
ivc	la_i_vc_set	irow, ivalue_r, ivalue_i
kvc	la_k_vc_set	krow, kvalue_r, kvalue_i
imr	la_i_mr_set	irow, icolumn, ivalue
kmr	la_k_mr_set	krow, kcolumn, ivalue
imc	la_i_mc_set	irow, icolumn, ivalue_r, ivalue_i
kmc	la_k_mc_set	krow, kcolumn, kvalue_r, kvalue_i
ivalue	la_i_get_vr	ivr, irow
kvalue	la_k_get_vr	ivr, krow
ivalue_r, ivalue_i	la_i_get_vc	ivc, irow
kvalue_r, kvalue_i	la_k_get_vc	ivc, krow
ivalue	la_i_get_mr	imr, irow, icolumn
kvalue	la_k_get_mr	imr, krow, kcolumn
ivalue_r, ivalue_i	la_i_get_mc	imc, irow, icolumn
kvalue_r, kvalue_i	la_k_get_mc	imc, krow, kcolumn

## Opérations sur un Tableau

imr	la_i_transpose_mr	imr
imr	la_k_transpose_mr	imr
imc	la_i_transpose_mc	imc
imc	la_k_transpose_mc	imc
ivr	la_i_conjugate_vr	ivr
ivr	la_k_conjugate_vr	ivr
ivc	la_i_conjugate_vc	ivc
ivc	la_k_conjugate_vc	ivc
imr	la_i_conjugate_mr	imr

imr	la_k_conjugate_mr	imr
imc	la_i_conjugate_mc	imc
imc	la_k_conjugate_mc	imc

## Opérations scalaires

ir	la_i_norm1_vr	ivr
kr	la_k_norm1_vr	ivc
ir	la_i_norm1_vc	ivc
kr	la_k_norm1_vc	ivc
ir	la_i_norm1_mr	imr
kr	la_k_norm1_mr	imr
ir	la_i_norm1_mc	imc
kr	la_k_norm1_mc	imc
ir	la_i_norm_euclid_vr	ivr
kr	la_k_norm_euclid_vr	ivr
ir	la_i_norm_euclid_vc	ivc
kr	la_k_norm_euclid_vc	ivc
ir	la_i_norm_euclid_mr	mvr
kr	la_k_norm_euclid_mr	mvr
ir	la_i_norm_euclid_mc	mvc
kr	la_k_norm_euclid_mc	mvc
ir	la_i_distance_vr	ivr
kr	la_k_distance_vr	ivr
ir	la_i_distance_vc	ivc
kr	la_k_distance_vc	ivc
ir	la_i_norm_max	imr
kr	la_k_norm_max	imc
ir	la_i_norm_max	imr

---



kr	la_k_norm_max	imc
ir	la_i_norm_inf_vr	ivr
kr	la_k_norm_inf_vr	ivr
ir	la_i_norm_inf_vc	ivc
kr	la_k_norm_inf_vc	ivc
ir	la_i_norm_inf_mr	imr
kr	la_k_norm_inf_mr	imr
ir	la_i_norm_inf_mc	imc
kr	la_k_norm_inf_mc	imc
ir	la_i_trace_mr	imr
kr	la_k_trace_mr	imr
ir, ii	la_i_trace_mc	imc
kr, ki	la_k_trace_mc	imc
ir	la_i_lu_det	imr
kr	la_k_lu_det	imr
ir	la_i_lu_det	imc
kr	la_k_lu_det	imc

## Opérations sur les Eléments entre Tableaux

ivr	la_i_add_vr	ivr_a, ivr_b
ivc	la_k_add_vc	ivc_a, ivc_b
imr	la_i_add_mr	imr_a, imr_b
imc	la_k_add_mc	imc_a, imc_b
ivr	la_i_subtract_vr	ivr_a, ivr_b
ivc	la_k_subtract_vc	ivc_a, ivc_b
imr	la_i_subtract_mr	imr_a, imr_b
imc	la_k_subtract_mc	imc_a, imc_b
ivr	la_i_multiply_vr	ivr_a, ivr_b

ivc	la_k_multiply_vc	ivc_a, ivc_b
imr	la_i_multiply_mr	imr_a, imr_b
imc	la_k_multiply_mc	imc_a, imc_b
ivr	la_i_divide_vr	ivr_a, ivr_b
ivc	la_k_divide_vc	ivc_a, ivc_b
imr	la_i_divide_mr	imr_a, imr_b
imc	la_k_divide_mc	imc_a, imc_b

## Produits Scalaires

ir	la_i_dot_vr	ivr_a, ivr_b
kr	la_k_dot_vr	ivr_a, ivr_b
ir, ii	la_i_dot_vc	ivc_a, ivc_b
kr, ki	la_k_dot_vc	ivc_a, ivc_b
imr	la_i_dot_mr	imr_a, imr_b
imr	la_k_dot_mr	imr_a, imr_b
imc	la_i_dot_mc	imc_a, imc_b
imc	la_k_dot_mc	imc_a, imc_b
ivr	la_i_dot_mr_vr	imr_a, ivr_b
ivr	la_k_dot_mr_vr	imr_a, ivr_b
ivc	la_i_dot_mc_vc	imc_a, ivc_b
ivc	la_k_dot_mc_vc	imc_a, ivc_b

## Inversion de Matrice

imr, icondition	la_i_invert_mr	imr
imr, kcondition	la_k_invert_mr	imr
imc, icondition	la_i_invert_mc	imc
imc, kcondition	la_k_invert_mc	imc

## Décompositions et Résolutions de Matrice

ivr	la_i_upper_solve_mr	imr [, j_l_diagonal]
-----	---------------------	----------------------

ivr	<b>la_k_upper_solve_mr</b>	imr [, j_1_diagonal]
ivc	<b>la_i_upper_solve_mc</b>	imc [, j_1_diagonal]
ivc	<b>la_k_upper_solve_mc</b>	imc [, j_1_diagonal]
ivr	<b>la_i_lower_solve_mr</b>	imr [, j_1_diagonal]
ivr	<b>la_k_lower_solve_mr</b>	imr [, j_1_diagonal]
ivc	<b>la_i_lower_solve_mc</b>	imc [, j_1_diagonal]
ivc	<b>la_k_lower_solve_mc</b>	imc [, j_1_diagonal]
imr, ivr_pivot, isize	<b>la_i_lu_factor_mr</b>	imr
imr, ivr_pivot, ksize	<b>la_k_lu_factor_mr</b>	imr
imc, ivr_pivot, isize	<b>la_i_lu_factor_mc</b>	imc
imc, ivr_pivot, ksize	<b>la_k_lu_factor_mc</b>	imc
ivr_x	<b>la_i_lu_solve_mr</b>	imr, ivr_b
ivr_x	<b>la_k_lu_solve_mr</b>	imr, ivr_b
ivc_x	<b>la_i_lu_solve_mc</b>	imc, ivc_b
ivc_x	<b>la_k_lu_solve_mc</b>	imc, ivc_b
imr_q, imr_r	<b>la_i_qr_factor_mr</b>	imr
imr_q, imr_r	<b>la_k_qr_factor_mr</b>	imr
imc_q, imc_r	<b>la_i_qr_factor_mc</b>	imc
imc_q, imc_r	<b>la_k_qr_factor_mc</b>	imc
ivr_eig_vals	<b>la_i_qr_eigen_mr</b>	imr, i_tolerance
ivr_eig_vals	<b>la_k_qr_eigen_mr</b>	imr, k_tolerance
ivr_eig_vals	<b>la_i_qr_eigen_mc</b>	imc, i_tolerance
ivr_eig_vals	<b>la_k_qr_eigen_mc</b>	imc, k_tolerance



### Avertissement

Une matrice doit être hermitienne si l'on veut calculer ses valeurs propres.

ivr_eig_vals, imr_eig_vecs	<b>la_i_qr_sym_eigen_mr</b>	imr, i_tolerance
ivr_eig_vals, imr_eig_vecs	<b>la_k_qr_sym_eigen_mr</b>	imr, k_tolerance

`ivc_eig_vals, imc_eig_vecs   la_i_qr_sym_eigen_mc   imc, i_tolerance`

`ivc_eig_vals, imc_eig_vecs   la_k_qr_sym_eigen_mc   imc, k_tolerance`

## Crédits

Michael Gogins

Nouveau dans la version 5.09 de Csound

---

# Conversion des Hauteurs

## Fonctions

Les opcodes qui effectuent les fonctions de hauteur communes sont :

- *cent*
- *cpsmidinn*
- *cpsoct*
- *cpspch*
- *octave*
- *octcps*
- *octmidinn*
- *octpch*
- *pchmidinn*
- *pchoct*
- *semitone*

## Opcodes de Hauteurs

Les opcodes qui effectuent les fonctions d'accordage sont :

- *cps2pch*
- *cpsxpch*
- *cpstun*
- *cpstuni*

---

# Support MIDI en Temps-Réel

Csound supporte les entrées et les sorties MIDI en temps réel, ainsi que les entrées depuis les fichiers MIDI. L'entrée MIDI en temps réel est activée au moyen de l'option de ligne de commande `-M` (ou `--midi-device=PERIPHERIQUE`). Vous devez spécifier le numéro ou le nom de périphérique après le `-M`. Par exemple, pour utiliser le périphérique numéro 2, vous utiliserez quelque chose comme :

```
csound -M2 monmiditr.csd
```

Vous pouvez trouver les périphériques disponibles en utilisant un numéro trop grand :

```
csound -M99 monmiditr.csd
```



## Note

Ceci ne fonctionnera que si le module MIDI peut être atteint par numéro de périphérique. Pour alsa, il faut d'abord trouver le nom du périphérique en utilisant :

```
cat /proc/asound/cards
```

Il faut alors taper quelque chose comme :

```
csound --rtmidi=alsa -M hw:3 monmiditr.csd
```

La sortie MIDI en temps réel est activée au moyen de `-Q`, avec un numéro ou un nom de périphérique comme c'est montré ci-dessus.

Vous pouvez aussi charger un fichier MIDI en utilisant l'option de ligne de commande `-F` ou `--midifile=FICHIER`. Le fichier MIDI est lu en temps réel, et se comporte comme s'il était joué ou reçu en temps réel. Ainsi le programme csound ne sait pas si l'entrée MIDI vient d'un fichier MIDI ou directement d'une interface MIDI.

Une fois l'entrée et/ou la sortie MIDI activée(s), les opcodes comme *MIDI Input* et *MIDI Output* seront effectifs.

Quand l'entrée MIDI est activée (avec `-M` ou `-F`), chaque message de *noteon* entrant générera un événement de note pour un instrument qui a le même numéro que le canal de l'évènement (voir *massign* et *pgmassign* pour changer ce comportement). Cela signifie que les instruments contrôlés par le MIDI sont polyphoniques par défaut, car chaque note générera une nouvelle instance de l'instrument.

Voir les opcodes pour l'*Interopérabilité MIDI/Partition* pour savoir comment concevoir des instruments utilisables depuis une partition ou pilotés par le MIDI.

Plusieurs modules MIDI en temps réel sont disponibles, et il faut utiliser l'option `--rtmidi` (voir `-+rtmidi`), pour spécifier le module. Le module par défaut est *portmidi* qui fournit des E/S MIDI adéquates sur toutes les plates-formes, cependant, pour des performances améliorées et plus fiables, des modules spécifiques à certaines plates-formes sont également fournis.

Actuellement les modules midi disponibles sont :

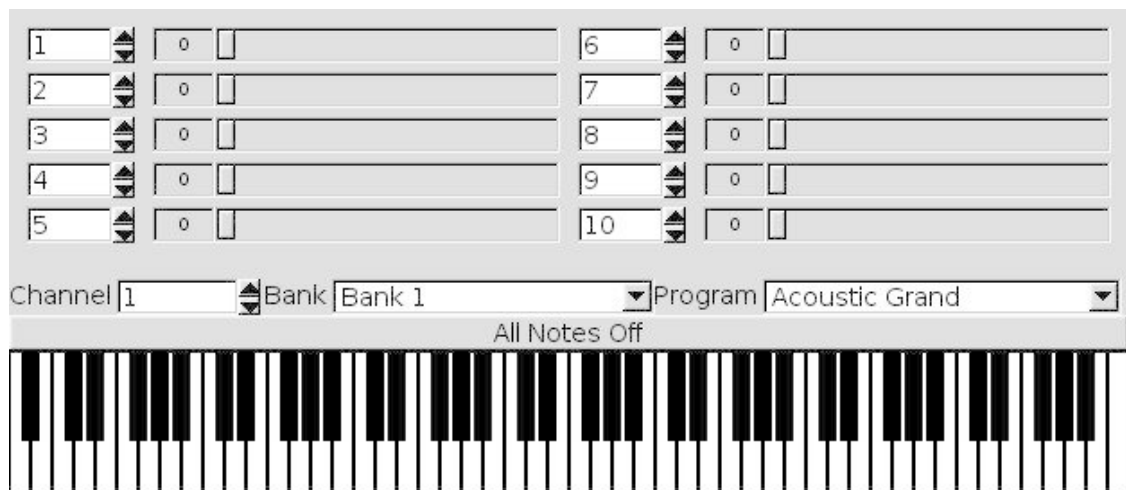
- *alsa* - Pour utiliser le système midi ALSA (seulement sur Linux)
- *winmm* - Pour utiliser le système windows MME (seulement sur Windows)
- *portmidi* - Pour utiliser le système portmidi (sur toutes les plates-formes). C'est le réglage par défaut.
- *virtual* - Pour utiliser un clavier virtuel graphique (voir ci-dessous) comme entrée MIDI (sur toutes les plates-formes)



### Astuce

Lors de son exécution, Csound traite la partition puis se termine. S'il n'y a pas d'évènements dans la partition, Csound se termine immédiatement. Si l'on désire n'utiliser que des évènements MIDI au lieu des évènements de partition, il faut demander à Csound de s'exécuter pendant un certain temps au moyen d'une *instruction f* comme "f 0 3600".

## Clavier Virtuel MIDI



Clavier Virtuel MIDI.

Le module du clavier virtuel MIDI (activé par l'option `-+rtmidi=virtual` sur la ligne de commande) fournit un moyen d'envoyer des informations MIDI en temps réel à Csound sans avoir besoin d'un périphérique MIDI. Il peut envoyer des informations de note, des changements de contrôle, des changements de banque et de programme sur un canal spécifié. L'information MIDI en provenance du clavier virtuel est traitée par Csound exactement de la même manière que si elle venait d'autres pilotes MIDI, si bien que si votre orchestre Csound est conçu pour travailler avec des périphériques matériels MIDI, cela marchera aussi.

Le clavier virtuel utilise l'option de périphérique (`-M`) pour récupérer le nom d'un fichier de mappage du clavier. Comme tous les pilotes MIDI, celui-ci nécessite un périphérique pour être activé. Si l'on désire seulement utiliser les réglages par défaut du clavier, il suffit de passer 0 (c'est-à-dire `-M0`). Si au lieu de 0 un nom de fichier est donné, le clavier essaiera de charger le fichier pour le mappage du clavier. Si le fichier n'a pas pu être ouvert ou lu correctement, les réglages par défaut seront utilisés.

Les fichiers de Mappage du Clavier permettent à l'utilisateur de personnaliser le nom et le numéro des

banques ainsi que le nom et le numéro des programmes d'une banque. L'exemple suivant de mappage de clavier (nommé `keyboard.map`) a des commentaires intégrés sur le format de fichier. Ce fichier est aussi disponible dans la distribution des sources de Csound dans le répertoire `InOut/virtual_keyboard`.

```
# Carte de Personnalisation du Clavier pour le Clavier Virtuel
# Steven Yi
#
# USAGE
#
# Lors de l'utilisation du clavier virtuel, vous pouvez fournir un nom de fichier
# pour un mappage des banques et des programmes via l'option -M, par exemple :
#
# csound -+rtmidi=virtual -Mkeyboard.map mon_projet.csd
#
# INFORMATION SUR LE FORMAT
#
# -les lignes commençant par '#' sont des commentaires
# -les lignes avec [] commencent les définitions d'une nouvelle banque,
# les contenus sont numBanque=nomBanque, avec numBanque=[1,16384]
# -les lignes suivant les instructions de banque sont des définitions de programme
# dans le format numProgramme=nomProgramme, avec numProgramme=[1,128]
# -les numéros de banque et de programme sont définis dans ce fichier
# en commençant à 1, mais ils sont convertis en valeurs midi (commençant
# à 0) lorsqu'ils sont lus
#
# NOTES
#
# -si une définition de banque invalide est trouvée, toutes les
# définitions de programme qui suivent seront ignorées jusqu'à ce
# qu'une nouvelle définition de banque valide soit trouvée
# -si une définition valide de banque sans programmes valides est
# trouvée, elle prendra par défaut les définitions de programme
# General MIDI
# -si une définition de programme invalide est trouvée, elle sera
# ignorée

[1=Ma Banque]
1=Mon Patch de Test 1
2=Mon Patch de Test 2
30=Mon Patch de Test 30

[2=Ma Banque2]
1=Mon Patch de Test 1(banque2)
2=Mon Patch de Test 2(banque2)
30=Mon Patch de Test 30(banque2)
```

Les dix réglettes du haut sont affectées par défaut aux contrôleurs MIDI numéro 1-10, mais on peut les changer à volonté. Les numéros de contrôleur et les valeurs de chaque réglette sont fixés par canal, si bien que l'on peut utiliser différents réglages et valeurs pour chaque canal.

Par défaut il y a 128 banques et pour chaque banque 128 patches réglés par défaut sur les noms General Midi. Le standard de banque MIDI utilise une résolution sur 14 bit pour supporter 16384 banques possibles, mais les numéros de banque par défaut sont 0-127. Pour utiliser des valeurs supérieures à 127, il faut utiliser un mappage de clavier personnalisé et fixer la valeur du numéro de banque désiré pour le nom de la banque. Le clavier virtuel transmettra correctement le numéro de banque comme MSB et LSB avec les contrôleurs 0 et 32.

Outre l'entrée disponible par l'interaction avec la GUI via la souris, on peut aussi déclencher les notes MIDI à partir du clavier ASCII quand la fenêtre du clavier virtuel a le focus. L'arrangement est organisé à la manière d'un traceur et offre deux octaves et une tierce majeure, en partant du do médiant (note MIDI 60). La correspondance entre le clavier ASCII et les valeurs de note MIDI est donnée dans la table suivante.

**Tableau 6. Valeurs des Notes MIDI du Clavier ASCII**

Touche	Valeur MIDI
z	60



Touche	Valeur MIDI
s	61
x	62
d	63
c	64
v	65
g	66
b	67
h	68
n	69
j	70
m	71
q	72
2	73
w	74
3	75
e	76
r	77
5	78
t	79
6	80
y	81
7	82
u	83
i	84
9	85
o	86
0	87
p	88

Voici un exemple de l'utilisation du clavier MIDI virtuel. Il utilise le fichier *virtual.csd* [exemples/virtual.csd].

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Virtual MIDI  -M0 is needed anyway
-odac          -iadc    --rtmidi=virtual -M0
</CsOptions>
<CsInstruments>
; By Mark Jamerson 2007

sr=44100
ksmps=10
nchnls=2

massign 1,1
prealloc 1,10

instr 1 ;Midi FM synth
```

```
inote cpsmidi
iveloc ampmidi 10000
idur = 2
    xtratim 1

kgate oscil 1,10,2
anoise noise 100*inote,.99
acps samphold anoise,kgate
aosc oscili 1000,acps,1
aout = aosc

; Use controller 7 to control volume
kvol ctrl7 1, 7, 0.2, 1

outs kvol * aout, kvol * aout

endin

</CsInstruments>
<CsScore>
f0 3600
f1 0 1024 10 1
f2 0 16 7 1 8 0 8
f3 0 1024 10 1 .5 .6 .3 .2 .5

e
</CsScore>
</CsoundSynthesizer>
```

## Entrée MIDI

Les opcodes suivants peuvent recevoir des informations MIDI :

- Information MIDI pour tous les instruments : *aftouch*, *chanctrl* et *polyaft*, *pchbend*.
- Information MIDI pour les instruments déclenchés par le MIDI : *veloc*, *midictrl* et *notnum*. Voir aussi *Converters*.
- Entrée de Contrôleur MIDI pour tous les instruments : *ctrl7*, *ctrl14* et *ctrl21*.
- Entrée de Contrôleur MIDI seulement pour les instruments déclenchés par le MIDI : *midic7*, *midic14* et *midic21*.
- Valeur d'initialisation de contrôleur MIDI : *initc7*, *initc14*, *initc21* et *ctrlinit*.

*massign* peut être utilisé pour spécifier l'instrument csound à déclencher par un canal MIDI particulier.  
*pgmassign* peut être utilisé pour assigner un instrument csound à un programme MIDI spécifique.

## Sortie de Message MIDI

Les opcodes qui produisent des sorties MIDI sont :

- *mdelay*
- *nrpn*
- *outiat*
- *outic*
- *outic14*

- *outipat*
- *outipb*
- *outipc*
- *outkat*
- *outkc*
- *outkc14*
- *outkpat*
- *outkpb*
- *outkpc*

## Entrée et Sortie Génériques

Les opcodes pour les entrées et les sorties MIDI génériques sont : *midiin* et *midiout*.

## Convertisseurs

Les opcodes suivants peuvent convertir de l'information MIDI provenant d'une instance d'un instrument déclenché par le MIDI :

- Convertisseurs de numéros de note MIDI en fréquence : *cpsmidi*, *cpsmidib*, *cpstmid*, *octmidi*, *octmidib*, *pchmidi* et *pchmidib*.
- Convertisseurs de vélocité MIDI en amplitude : *ampmidi*.

## Extension d'Evènements

Les opcodes qui permettent d'étendre la durée d'un évènement sont :

- *release*
- *xtratim*

## Sortie de Note-on/Note-off

Les opcodes pour sortir des messages MIDI noteon ou noteoff sont :

- *midion*
- *midion2*
- *moscil*

- *noteoff*
- *noteon*
- *noteondur*
- *noteondur2*

## Opcodes pour l'Interopérabilité MIDI/Partition

Les opcodes suivants peuvent être utilisés pour concevoir des instruments qui fonctionnent de manière interchangeable avec du MIDI en temps réel et avec des événements de partition :

- *midichannelaftertouch*
- *midichn*
- *midicontrolchange*
- *mididefault*
- *midinoteoff*
- *midinoteoncps*
- *midinoteonkey*
- *midinoteonoct*
- *midinoteonpch*
- *midipitchbend*
- *midipolyaftertouch*
- *midiprogramchange*.



### Adapter un instrument Csound déclenché par une partition.

Pour adapter un instrument Csound ordinaire conçu pour être activé depuis une partition, à l'interopérabilité partition/MIDI :

- Changer tous les opcodes *linen*, *linseg*, et *expseg* respectivement en *linenr*, *linsegr*, et *expsegr*, sauf pour une enveloppe de décliquage ou d'atténuation. Cela ne changera en rien les exécutions pilotées par une partition.
- Ajouter les lignes suivantes au début de la définition de l'instrument :

```
; Pour être sûr qu'un instrument activé par le MIDI
; aura un champ p3 positif.
mididefault 60, p3
; Met le numéro de touche MIDI traduit en cycles par
; seconde dans p4, et la vélocité MIDI dans p5
midinoteoncps p4, p5
```

Bien entendu, *midinoteoncps* pourrait être changé en *midinoteonoct* ou tout autre option, et le choix des p-champs est arbitraire.



## Options de ligne de commande d'Entrée/Sortie MIDI en Temps Réel

Les nouvelles *options d'E/S MIDI* dans Csound 5.02, peuvent remplacer la plupart des utilisations de ces opcodes d'interopérabilité, et en rendre l'usage plus facile.

## Messages System Realtime

Les opcodes pour les messages MIDI System Realtime sont : *mclock* et *mrtmsg*.

## Banques de Réglettes

Les opcodes pour les banques de réglettes de contrôleurs MIDI sont :

- *slider8*
- *slider8f*
- *slider16*
- *slider16f*
- *slider32*
- *slider32f*
- *slider64*
- *slider64f*
- *s16b14*
- *s32b14*
- *sliderKawai*

Les opcodes pour stocker des banques de réglettes de contrôleurs MIDI dans des tables sont :

- *slider8table*
- *slider8tablef*
- *slider16table*
- *slider16tablef*
- *slider32table*

- *slider32tablef*
- *slider64table*
- *slider64tablef*

---

# Traitement Spectral

voir la section *Synthèse/Resynthèse Additive* pour les opcodes élémentaires de resynthèse.

## Resynthèse par Transformée de Fourier à Court-Terme (STFT)



### Utilisation des fichiers PVOC-EX avec les anciens opcodes pvoc de Csound

Tous les opcodes pvoc originaux peuvent lire maintenant des fichiers PVOC-EX, aussi bien que le format de fichier natif non portable. Comme un fichier PVOC-EX utilise une fenêtre d'analyse de taille double, les utilisateurs trouveront sans doute que le résultat est utilement amélioré, pour certains sons et certains traitements, malgré le fait que la resynthèse n'utilise pas la même taille de fenêtre.

En dehors du paramètre de taille de fenêtre, la différence principale entre le format original .pv et PVOC-EX est l'intervalle d'amplitude des trames d'analyse. Lorsque la pondération est appliquée, afin qu'il n'y ait pas de différences notables dans le niveau de sortie, quelque soit le format de fichier utilisé, de légères pertes d'amplitude peuvent encore se produire, car l'utilisation d'une fenêtre double modifie l'amplitude des trames, sans que le code de resynthèse en tienne compte. Noter que tous les opcodes pvoc originaux attendent un fichier d'analyse mono, et que les fichiers PVOC-EX multi-canaux seront ainsi rejetés.

Les opcodes qui implémentent la resynthèse STFT sont :

- *tableseg*
- *pvadd*
- *pvbufread*
- *pvcross*
- *pvinterp*
- *pvoc*
- *pvread*
- *tableseg*
- *tablexseg*
- *vpvoc*

L'utilitaire *PVANAL* permet de générer les fichiers d'analyse pv.

## Resynthèse par Codage Prédicatif Linéaire (LPC)

Les opcodes de resynthèse par prédiction linéaire sont :

- *lpfreson*
- *lpinterp*
- *lpread*
- *lpreson*
- *lpslot*

On peut créer des fichiers d'analyse LPC au moyen de l'utilitaire *LPANAL*.

## Traitement Spectral Non-standard

Ces unités génèrent et traitent des types de données de signaux non-standard, tels que des signaux de contrôle du domaine temporel et des signaux audio sous-échantillonnés, et leur représentation dans le domaine fréquentiel (spectrale). Les types de données (*d-*, *w-*) se définissent par eux-mêmes et leur contenu n'est pas utilisable par les autres unités de Csound. Ces générateurs unitaires sont expérimentaux, et sujets à modification entre les différentes versions de Csound ; ils seront aussi complétés par d'autres unités plus tard.

Les opcodes pour le traitement spectral non-standard sont *specaddm*, *specdiff*, *specdisp*, *specfilt*, *spechist*, *specptrk*, *specscal*, *specsum* et *spectrum*.

## Outils pour le Traitement Spectral en Temps Réel (opcodes pvs)

Avec ces opcodes, deux nouvelles facilités fondamentales sont ajoutées à Csound. Ils offrent une qualité audio améliorée, et une exécution rapide, permettant une analyse et une resynthèse de grande qualité (avec les transformations) à appliquer en temps réel aux signaux instantanés. Le vocodeur de phase original de Csound n'est pas changé ; les nouveaux opcodes utilisent un ensemble de fonctions complètement séparé basé sur « *pvoc.c* » dans la distribution CARL, écrite par Mark Dolson.

Les utilitaires de Csound *dnoise* et *srconv* (également par Dolson, de CARL) utilisent aussi ce moteur *pvoc*. *pvoc* de CARL est aussi la base pour le vocodeur de phase inclu dans le Composer's Desktop Project. Quelques petites modifications, mais importantes, ont été apportées au code CARL original pour supporter les flots de données en temps réel.

1. Support du nouveau format de fichier d'analyse PVOC-EX. C'est un format totalement portable et ouvert (multi plates-formes), supportant trois formats d'analyse, et les signaux multi-canaux. Actuellement seul le format standard amplitude+fréquence a été implémenté dans les opcodes, mais le format de fichier lui-même supporte les formats amplitude+phase et le format complexe (réel-imaginaire). En plus des nouveaux opcodes, les opcodes *pvoc* originaux de Csound ont été étendus (avec pour conséquence une qualité audio améliorée dans certains cas) pour lire les fichiers PVOC-EX aussi bien que le format original (non portable).



Les détails complets de la structure d'un fichier PVOC-EX son disponibles sur le site web : <http://www.cs.bath.ac.uk/~jpff/NOS-DREAM/researchdev/pvocex/pvocex.html>. Ce site donne aussi les détails des programmes de console disponibles librement *pvocex* et *pvocex2* qui peuvent être utilisés pour créer des fichiers PVOC-EX dans tous les formats supportés.

2. Un nouveau type de signal du domaine fréquentiel, totalement transportable par flot de données, avec *f* comme premier caractère. Dans ce document on y fait référence par *fsig*. Le support principal des *fsigs* est fourni par les opcodes *pvsanal* et *pvsynth*, qui effectuent l'analyse et la resynthèse traditionnelles par chevauchement-addition avec un vocodeur de phase, indépendamment du taux de contrôle de l'orchestre. La seule obligation est que le taux de contrôle *kr* soit supérieur ou égal au taux d'analyse, ce qui peut s'exprimer par  $ksmps \leq overlap$ , où *overlap* est la distance en échantillons entre deux trames d'analyse, comme spécifié pour *pvsanal*. Comme *overlap* vaut typiquement au moins 128, et plus souvent 256, ce n'est pas une restriction coûteuse en pratique. L'opcode *pvsinfo* peut être utilisé au moment de l'initialisation pour acquérir les propriétés d'un *fsig*.

Le *fsig* permet la séparation nominale entre les étapes d'analyse et de resynthèse du vocodeur de phase pour une mise à disposition du programmeur Csound, ce qui permet non seulement d'employer des alternatives pour l'une ou les deux de ces étapes (pas seulement la resynthèse par banc d'oscillateur, mais aussi la génération synthétique de flots de données *fsig*), mais aussi les opcodes opérant sur le flot *fsig* peuvent être eux-mêmes plus élémentaires. Ainsi le *fsig* permet la création d'un véritable environnement de plugin de flots de données pour les signaux du domaine fréquentiel. Avec les vieux opcodes *pvoc*, chaque opcode doit pouvoir agir comme un resynthétiseur, si bien que des facilités comme la transposition de hauteur sont dupliquées dans chaque opcode ; et dans la plupart des cas les opcodes ont beaucoup de paramètres. La séparation des étapes d'analyse et de synthèse au moyen du *fsig* encourage le développement d'une grande variété d'opcodes qui sont des briques élémentaires implémentant une ou deux fonctions, et avec lesquelles on peut construire des processus plus élaborés.

Cette réalisation en est encore à ses débuts et présente un caractère expérimental, et il est possible que la définition précise des opcodes change en réponse aux avis des utilisateurs. De plus, de nombreuses nouvelles possibilités d'opcode sont ouvertes ; ces facteurs peuvent aussi avoir une influence rétrospective sur les opcodes présentés ici.

Noter que certains paramètres d'opcode ont actuellement une implémentation restreinte ou manquante. Ceci, au moins en partie, afin de préserver la simplicité des opcodes à ce niveau, et aussi parce qu'ils concernent d'importantes questions de conception pour lesquelles aucune décision n'a encore été prise, et pour lesquelles l'opinion des utilisateurs est souhaitée.

Un point important au sujet de ce nouveau type de signal est que, parce que le taux d'analyse est typiquement très inférieur à *kr*, les nouvelles trames d'analyse ne sont pas disponibles à chaque *k*-cycle. En interne, les opcodes tracent *ksmps*, et maintiennent également un compteur de trames, afin que les trames soient lues et écrites aux bons moments ; ce processus est généralement transparent pour l'utilisateur. Cependant, cela signifie que les signaux de taux-*k* n'agissent sur un *fsig* qu'au taux d'analyse, pas à chaque *k*-cycle. L'opcode *pvsftw* retourne un drapeau au taux-*k* qui est positionné lorsque de nouvelles données *fsig* sont disponibles.

A cause de la nature du système de chevauchement-addition, l'utilisation des ces opcodes infère un délai, ou latence, petit mais significatif déterminé par la taille de la fenêtre ( $\max(\text{ifftsize}, \text{iwinsize})$ ). Il vaut typiquement 23ms. Dans cette première réalisation, le délai dépasse légèrement le minimum théorique, et l'on espère qu'il pourra être réduit, lorsque les opcodes seront optimisés pour le transport par flot de données en temps-réel.

Les opcodes pour le traitement spectral en temps réel sont *pvsadsyn*, *pvsanal*, *pvsfread*, *pvsftr*, *pvsftw*, *pvsinfo*, *pvsmaska* et *pvsynth*.

De plus il y a un certain nombre d'opcodes disponibles sous forme de plugins dans Csound 5. Ce sont

*pvsdiskin, pvscent, pvsdemix, pvsfreeze, pvsbuffer, pvsbufread, pvscale, pvshift, pvsifd, pvsinit, pvsin, pvsout, pvsosc, pvsbin, pvdisp, pvfwrite, pvmix, pvsmooth, pvfilter, pvblur, pvstencil, pvsarp, pvsvoc, pvmorph, pvsbandp, pvsbandr*

Un certain nombre d'opcodes sont conçus pour générer et traiter des flots de données de pistes de partiels. Ce sont *partials, trcross, trfilter, trsplit, trmix, trscale, trshift, trlowest, trhighest, tradsyn, sinsyn, resyn, binit*

Voir la section *Piles* pour une information sur les opcodes qui peuvent empiler les signaux de type f.

## Traitement Spectral avec ATS

Ces opcodes peuvent lire, transformer et resynthétiser des fichiers d'analyse ATS. Prière de noter que l'application ATS est nécessaire pour produire les fichiers d'analyse. Voici un extrait du Manuel de Référence d'ATS.

*« ATS est une bibliothèque de fonctions pour l'Analyse spectrale, la Transformation et la Synthèse du son basée sur un modèle sinusoïdal plus du bruit de bande critique. Un son dans ATS est un objet symbolique représentant un modèle spectral qui peut être sculpté au moyen de diverses fonctions de transformation. »*

Pour plus d'information sur ATS visiter : <http://www-ccrma.stanford.edu/~juan/ATS.html>.

Les fichiers d'analyse ATS peuvent être produits avec le logiciel ATS ou l'utilitaire csound ATSA.

Les opcodes pour le traitement ATS sont :

- *ATSinfo* : lit les données de l'en-tête d'un fichier ATS.
- *ATSread, ATSreadnz, ATSbufread, ATSinterpread, ATSpartialtap* : lisent les données d'un fichier ou d'un tampon ATS.
- *ATSadd, ATSaddnz, ATScross, ATSSinnoi* : Resynthétisent le son.

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

## Opcodes Loris



### Note

Ces opcodes sont un composant facultatif de Csound5. Pour savoir s'ils sont installés utilisez la commande 'csound -z' qui donne la liste des opcodes disponibles.

La famille des opcodes Loris encapsule : *lorisread* qui importe un ensemble de partiels à largeur de bande adaptée depuis un fichier de données au format SDIF, en appliquant, au taux de contrôle, des enveloppes de pondération de fréquence, d'amplitude et de largeur de bande, et qui stocke les partiels modifiés en mémoire ; *lorismorph*, qui opère une transformation (morphing) entre deux ensembles stockés de partiels à largeur de bande adaptée et stocke un nouvel ensemble de partiels représentant le son transformé. La transformation est réalisée en interpolant linéairement les enveloppes des paramètres

(fréquence, amplitude, et largeur de bande, ou aspect bruiteux) des partiels à largeur de bande adaptée selon des fonctions de transformation de la fréquence, de l'amplitude et de la largeur de bande, variant au taux de contrôle ; *lorisplay*, qui restitue un ensemble de partiels à largeur de bande adaptée en utilisant la méthode de Synthèse Additive à Largeur de Bande Adaptée implémentée dans le logiciel Loris, avec application d'enveloppes de pondération de fréquence, d'amplitude, et de largeur de bande, variant au taux de contrôle.

Pour plus d'information sur la transformation du son et sa manipulation avec Loris et le Modèle Additif à Largeur de Bande Adaptée Réassignée, visiter le site web de Loris à [www.cerlsoundgroup.org/Loris](http://www.cerlsoundgroup.org/Loris) [<http://www.cerlsoundgroup.org/Loris>].

## Exemples

### Exemple 3. Jouer les partiels sans modification

```
;
; Joue les partiels dans clarinet.sdif
; de 0 à 3 sec avec un temps de transition de 1 ms
; et sans modification de fréquence, d'amplitude,
; ou de largeur de bande.
;
instr 1
  ktime    linseg      0, p3, 3.0    ; fonction linéaire du temps de 0 à 3 secondes
           lorisread   ktime, "clarinet.sdif", 1, 1, 1, 1, .001
  asig     lorisplay   1, 1, 1, 1
           out         asig
endin
```

### Exemple 4. Ajouter une intonation et un vibrato

```
; Joue les partiels dans clarinet.sdif
; de 0 à 3 sec avec un temps de transition de 1 ms
; ajout d'une intonation et d'un vibrato, accroissement
; du "souffle" (aspect bruiteux) et de l'amplitude
; générale, et ajout d'un filtre passe-haut.
;
instr 2
  ktime    linseg      0, p3, 3.0    ; fonction linéaire du temps de 0 à 3 secondes

  ; calcule le rapport de fréquence pour l'intonation
  ; (la hauteur originale était sol#3)
  ifscale  =          cpspch(p4)/cpspch(8.08)

  ; faire une enveloppe de vibrato
  kenv     linseg      0, p3/6, 0, p3/6, .02, p3/3, .02, p3/6, 0, p3/6, 0
  kvib     oscil       kenv, 4, 1    ; table 1, sinusoid

  kbwenv   linseg      1, p3/6, 1, p3/6, 2, 2*p3/3, 2
  lorisread ktime, "clarinet.sdif", 1, 1, 1, 1, .001
  a1       lorisplay   1, ifscale+kvib, 2, kbwenv
  a2       atone       a1, 1000      ; filtre passe-haut, coupure à 1000 Hz
           out         a2
endin
```

L'instrument du premier exemple synthétise un son de clarinette en utilisant du début à la fin les partiels dérivés de l'analyse à bande adaptée réassignée d'un son de clarinette de trois secondes, stockés dans le fichier `clarinet.sdif`. L'instrument de l'exemple 2 ajoute une intonation et un vibrato au son de clarinette synthétisé par l'instrument 1, renforce son amplitude et son aspect bruiteux, et applique un filtre passe-haut au résultat. La partition suivante peut être utilisée pour tester les deux instruments décrits ci-dessus.

```

; créer une sinus dans la table 1
f 1 0 4096 10 1

; jouer instr 1
;   début   dur
i 1 0       3
i 1 +       1
i 1 +       6
s

; jouer instr 2
;   début   dur   hauteur
i 2 1       3     8.08
i 2 3.5     1     8.04
i 2 4       6     8.00
i 2 4       6     8.07

e

```

### Exemple 5. Transformation de partiels

```

; Transforme les partiels de clarinet.sdif vers
; les partiels de flute.sdif sur la durée de la
; partie tenue des deux notes (de 0,2 à 2,0 secondes
; pour la clarinette, et de 0,5 à 2,1 secondes
; pour la flûte). Les portions d'attaque et de
; chute dans le son transformé sont spécifiées
; par les paramètres p4 et p5, respectivement.
; Le temps de transformation est le temps entre
; l'attaque et la chute. Les partiels de la
; clarinette sont transposés pour s'accorder à
; la hauteur de la note de la flûte (ré au-dessus
; du do médium).
;
instr 1
  ionset = p4
  idecay = p5
  itmorph = p3 - (ionset + idecay)
  ipshift = cpspch(8.02)/cpspch(8.08)

  ; fonction temporelle de la clarinette, transformation de 0,2 à 2,0 secondes
  ktcl linseg 0, ionset, .2, itmorph, 2.0, idecay, 2.1
  ; fonction temporelle de la flûte, transformation de 0,5 à 2,1 secondes
  ktfl linseg 0, ionset, .5, itmorph, 2.1, idecay, 2.3
  kmurph linseg 0, ionset, 0, itmorph, 1, idecay, 1
  lorisread ktcl, "clarinet.sdif", 1, ipshift, 2, 1, .001
  lorisread ktfl, "flute.sdif", 2, 1, 1, 1, .001
  lorismorph 1, 2, 3, kmurph, kmurph, kmurph
  asig lorisplay 3, 1, 1, 1
  out asig
endin

```

### Exemple 6. Plus de transformation

```

; Transforme les partiels de trombone.sdif vers les
; partiels de meow.sdif. Les dates de début et de fin
; de la transformation sont spécifiées par les
; paramètres p4 et p5, respectivement. La transformation
; a lieu sur la deuxième des quatre notes dans chaque
; son, de 0,75 à 1,2 secondes pour le trombone flatterzung,
; et de 1,7 à 2,2 secondes pour le miaulement de chat.
; Des fonctions de transformation différentes sont
; utilisées pour les enveloppes de fréquence et
; d'amplitude, afin que l'amplitude des partiels
; ait une transition plus rapide du trombone au
; chat que les fréquences. (Les enveloppes de largeur
; de bande utilisent la même fonction de transformation

```

```

; que les amplitudes).
;
instr 2
  ionset = p4
  imorph = p5 - p4
  irelease = p3 - p5

  ktbn linseg 0, ionset, .75, imorph, 1.2, irelease, 2.4
  ktmeow linseg 0, ionset, 1.7, imorph, 2.2, irelease, 3.4

  kmfreq linseg 0, ionset, 0, .75*imorph, .25, .25*imorph, 1, irelease, 1
  kmamp linseg 0, ionset, 0, .75*imorph, .9, .25*imorph, 1, irelease, 1

  lorisread ktbn, "trombone.sdif", 1, 1, 1, 1, .001
  lorisread ktmeow, "meow.sdif", 2, 1, 1, 1, .001
  lorismorph 1, 2, 3, kmfreq, kmamp, kmamp
  lorisplay 3, 1, 1, 1
  out asig
endin

```

L'instrument dans le premier exemple effectue une transformation du son entre une note de clarinette et une note de flûte en utilisant les partiels à bande adaptée réassignée stockés dans `clarinet.sdif` et dans `flute.sdif`.

La transformation est effectuée sur les portions tenues des notes, 0,2 à 2,0 secondes dans le cas de la note de clarinette et 0,5 à 2,1 secondes dans le cas de la note de flûte. Les fonctions d'index temporel, `ktcl` et `ktfl`, alignent les portions d'attaque et de chute des notes avec les temps d'attaque et de chute du son transformé, spécifiées respectivement par les paramètres `p4` et `p5`. L'attaque du son transformé est entièrement composée des données de partiel de la clarinette, et la chute est entièrement composée de données de la flûte. Les partiels de la clarinette sont transposés pour s'accorder à la hauteur de la note de flûte (ré au-dessus du do médium).

L'instrument dans le second exemple effectue une transformation du son entre une note de trombone *flutterzung* et un miaulement de chat en utilisant les partiels à bande adaptée réassignée stockés dans `trombone.sdif` et `meow.sdif`. Les données dans ces fichiers SDIF ont été réparties par canaux et séparées pour établir une correspondance entre partiels.

Les deux ensembles de partiels sont importés et stockés dans des positions mémoire étiquetées 1 et 2, respectivement. Les deux sons originaux ont quatre notes, et la transformation est effectuée sur la seconde note de chaque son (de 0,75 à 1,2 secondes pour le trombone *flutterzung*, et de 1,7 à 2,2 secondes pour le miaulement de chat). Les fonctions d'index temporel, `ktbn` et `ktmeow`, alignent ces segments des ensembles de partiels source et cible avec les paramètres spécifiés pour la durée du début, de la fin, et totale de la transformation. Deux fonctions de transformation différentes sont utilisées, afin que les amplitudes des partiels et les coefficients de largeur de bande se transforment rapidement des valeurs du trombone aux valeurs du miaulement de chat, tandis que les fréquences opèrent une transition plus graduelle. Les partiels transformés sont stockés dans la position mémoire étiquetée 3 et restitués par l'instruction *lorisplay* qui suit. Ils auraient pu aussi être utilisés comme source pour une autre transformation dans un instrument de transformation à trois étapes. La partition suivante peut être utilisée pour tester les deux instruments décrits ci-dessus.

```

; jouer instr 1
;   début   dur   attaque   chute
i 1  0      3     .25      .15
i 1  +      1     .10      .10
i 1  +      6     1.       1.
s

; jouer instr 2
;   début   dur   début_morph   fin_morph
i 2  0      4     .75          2.75
e

```

## Crédits

Cette implémentation des générateurs unitaires Loris a été écrite par Kelly Fitz ([loris@cerlsoundgroup.org](mailto:loris@cerlsoundgroup.org) [<mailto:loris@cerlsoundgroup.org>]).

Elle est construite d'après une implémentation prototype du générateur unitaire *lorisplay* écrite par Corbin Champion, et basée sur la méthode de Synthèse Additive à Largeur de Bande Adaptée et sur les algorithmes de transformation du son implémentés dans la bibliothèque Loris pour la modélisation et la manipulation du son. Les opcodes ont été ensuite adaptés en plugin pour Csound 5 par Michael Gogins.

---

# Chaînes de Caractères

Les variables chaîne de caractères sont des variables dont le nom commence par S ou par gS (pour les variables chaîne locales ou globales, respectivement), et elle peuvent mémoriser n'importe quelle chaîne avec une longueur maximale définie par l'option de ligne de commande `++max_str_len` (255 caractères par défaut). On peut utiliser ces variables comme argument d'entrée de n'importe quel opcode qui attend une chaîne constante entre apostrophes, et on peut les manipuler durant les périodes d'initialisation ou d'exécution avec les opcodes dont la liste suit.

Il est également possible d'utiliser des chaînes dans les p-champs. Un p-champ chaîne peut être utilisé directement par plusieurs opcodes de l'orchestre, ou il peut être d'abord copié dans une variable chaîne :

```
a1      diskin2 p5, 1
```

```
Snom    strget p5  
a1      diskin2 Snom, 1
```

Les chaînes dans Csound peuvent être exprimées par les doubles apostrophes traditionnelles (" "), mais aussi par {{ }}. La seconde méthode est utile si l'on veut utiliser les caractères ';' et '\$' dans la chaîne sans avoir recours aux codes ASCII.



## Note

Les variables chaînes et les opcodes correspondants ne sont pas disponibles dans les versions de Csound antérieures à la 5.00.

On peut également lier une chaîne à un numéro au moyen de *strset* et *strget*.

Csound 5 a aussi amélioré l'analyse des constantes chaîne. Il est possible de spécifier une chaîne multi-lignes en l'entourant avec {{ et }} à la place des habituelles doubles apostrophes (noter que la longueur des constantes chaîne n'est pas limitée, et n'est pas affectée par l'option `++max_str_len`), et les séquences d'échappement suivantes sont automatiquement converties :

- \a : cloche d'alerte
- \b : retour arrière
- \n : nouvelle ligne
- \r : retour chariot
- \t : tabulation
- \\ : le caractère '\'
- \nnn : le caractère ayant le code ASCII (en octal) nnn

Les chaînes peuvent être utilement employées avec l'opcode *system* :

```
instr 1  
; csound5 permet de placer une chaîne sur plusieurs lignes dans des accolades doubles
```

```
system {{      ps
          date
          cd ~/Desktop
          pwd
          ls -l
          whois csounds.com
      }}
endin
```

Et avec les *opcodes python*, entre autres :

```
pyruni {{
import random

pool = [(1 + i/10.0) ** 1.2 for i in range(100)]

def get_number_from_pool(n, p):
    if random.random() < p:
        i = int(random.random() * len(pool))
        pool[i] = n
    return random.choice(pool)
}}
```

## Opcodes de Manipulation de Chaîne

Ces opcodes effectuent des opérations sur les variables chaîne (note : la plupart des opcodes ne sont exécutés qu'au moment de l'initialisation, et ils ont une version avec un suffixe "k" qui s'exécute au taux-i et au taux-k ; les exceptions à cette règle comprennent *puts* et *strget*) :

- *strcpy* et *strcpyk* - Assignment à une variable chaîne.
- *strcat* et *strcatk* - Concaténation de chaînes, avec mémorisation du résultat dans une variable.
- *strcmp* et *strcmpk* - Comparaison de chaînes.
- *strget* - Assignment à une variable chaîne de la valeur trouvée dans la table *strset* à l'index spécifié, ou d'un p-champ chaîne de la partition.
- *strlen* et *strlenk* - Retourne la longueur d'une chaîne.
- *sprintf* - conversion de sortie formatée à la manière de *printf*, avec mémorisation du résultat dans une variable chaîne.
- *sprintfk* - conversion de sortie formatée à la manière de *printf*, avec mémorisation du résultat dans une variable chaîne au taux-k.
- *puts* - Impression d'une constante ou d'une variable chaîne.
- *strindex* et *strindexk* - Retourne la première occurrence d'une chaîne dans une autre chaîne.
- *strrindex* et *strrindexk* - Retourne la dernière occurrence d'une chaîne dans une autre chaîne.
- *strsub* et *strsubk* - Retourne une sous-chaîne de la chaîne passée en paramètre.

## Opcodes de Conversion de Chaîne

Ces opcodes convertissent des variables chaînes (note : la plupart des opcodes ne sont exécutés qu'au moment de l'initialisation, et ils ont une version avec un suffixe "k" qui s'exécute au taux-i et au taux-k ; les exceptions à cette règle comprennent *puts* et *strget*) :



- *strtod* et *strtodk* - Convertit une valeur de chaîne en une valeur en virgule flottante.
- *strtol* et *strtolk* - Convertit une valeur de chaîne en un entier signé.
- *strchar* et *strchark* - Retourne le code ASCII d'un caractère dans une chaîne.
- *strlower* et *strlowerk* - Convertit une chaîne en minuscules.
- *strupper* et *strupperk* - Convertit une chaîne en majuscules.

---

# Opcodes Vectoriels

La famille des opcodes vectoriels est conçue pour pouvoir traiter des sections de f-table comme des vecteurs pour diverses opérations sur celles-ci.

## Opérateurs de Tableaux de Vecteurs

Les opcodes vectoriels suivants supportent les accès en lecture/écriture sur des tableaux de vecteurs (tableaux de tableaux) :

- *vtablei*
- *vtablelk*
- *vtablek*
- *vtablea*
- *vtablewi*
- *vtablewk*
- *vtablewa*
- *vtabi*
- *vtabk*
- *vtaba*
- *vtabwi*
- *vtabwk*
- *vtabwa*

## Opérations Entre un Signal Vectoriel et un Signal Scalaire

Ces opcodes effectuent des opérations numériques entre un signal de contrôle vectoriel (contenu dans une f-table), et un signal scalaire. Le résultat est un nouveau vecteur qui remplace les anciennes valeurs de la table. Il y a des versions de ces opcodes de taux-k et de taux-i.

Tous ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

Opérations Entre un Signal Vectoriel et un Signal Scalaire :

- *vadd*
- *vmult*

- *vpow*
- *vexp*
- *vadd\_i*
- *vmult\_i*
- *vpow\_i*
- *vexp\_i*

## Opérations Entre deux Signaux Vectoriels

Ces opcodes effectuent des opérations entre deux vecteurs, de telle manière que chaque élément du premier vecteur est traité avec l'élément correspondant de l'autre vecteur. Le résultat est un nouveau vecteur qui remplace les anciennes valeurs du vecteur source.

Opérations Entre deux Signaux Vectoriels :

- *vaddv*
- *vsbv*
- *vmultv*
- *vdivv*
- *vpowv*
- *vexpv*
- *vcopy*
- *vmap*
- *vaddv\_i*
- *vsbv\_i*
- *vmultv\_i*
- *vdivv\_i*
- *vpowv\_i*
- *vexpv\_i*
- *vcopy\_i*

Ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Générateurs Vectoriels d'Enveloppe

Les opcodes pour générer des vecteurs contenant des enveloppes sont *vlinseg* et *vexpseg*.

Ces opérateurs sont semblables à *linseg* et *expseg*, mais ils opèrent avec des signaux vectoriels à la place des signaux scalaires.

La sortie est un vecteur dans une f-table (préalablement allouée), tandis que chaque point charnière de l'enveloppe est en fait un vecteur de valeurs. Tous les points charnière doivent contenir le même nombre d'éléments (*ielements*).

Ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Limitation et Enroulement des Signaux Vectoriels de Contrôle

Les opcodes pour effectuer la limitation et l'enroulement des éléments dans un vecteur sont :

- *vlimit*
- *vwrap*
- *vmirror*

Ces opérateurs sont semblables à *limit*, *wrap* et *mirror*, mais ils opèrent sur un signal vectoriel à la place d'un signal scalaire. Les résultats remplacent les anciennes valeurs du vecteur contenues dans une f-table si celles-ci sont en dehors de l'intervalle min/max. Si l'on veut conserver le vecteur d'entrée, il faut utiliser l'opcode *vcopy* pour le copier dans une autre table.

Tous ces opcodes travaillent au taux-k.

Tous ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Chemins de Retard Vectoriel au Taux de Contrôle

Chemins de Retard Vectoriel au Taux de Contrôle :

- *vdelayk*
- *vport*
- *vecdelay*

## Générateurs de Signal Aléatoire Vectoriel

Ces opcodes génèrent des vecteurs de nombres aléatoires à stocker dans des tables. Ils génèrent une sorte de 'bruit vectoriel à bande limitée'. Tous ces opcodes fonctionnent au taux-k.

Générateurs de signal aléatoire vectoriel : *vrandh* et *vrandi*.

Des vecteurs d'automates cellulaires peuvent être générés au moyen de : *vcella*.

---

# Système de Patch Zak

Les opcodes zak sont utilisés pour créer un système de patch aux taux-i, -k et -a. On peut se représenter le système zak comme un tableau global de variables. Ces opcodes sont utiles pour réaliser de manière flexible des branchements et des routages d'un instrument à l'autre. Le système est semblable à une matrice de branchement sur une console de mixage ou à une matrice de modulation sur un synthétiseur. Il est aussi utile lorsque l'on a besoin d'un tableau de variables.

Le système zak est initialisé par l'opcode *zakinit* qui est habituellement placé juste après les autres initialisations globales : *sr*, *kr*, *ksmps*, *nchnls*. L'opcode *zakinit* définit deux plages de mémoire, une pour les patches aux taux-i et -k, et l'autre pour les patches au taux-a. L'opcode *zakinit* ne peut être appelé qu'une fois. Après l'initialisation de l'espace zak, on peut utiliser d'autres opcodes zak pour lire et écrire dans l'espace mémoire zak, ainsi qu'exécuter d'autres tâches.

Les opcodes zak sont comptés à partir de 0, si bien que si l'on définit un canal, le seul canal valide est le canal 0.

Les opcodes pour le système de patch zak sont :

- Taux Audio : *zacl*, *zakinit*, *zamod*, *zar*, *zarg*, *zaw* et *zawm*.
- Taux de Contrôle : *zkcl*, *zkmod*, *zkr*, *zkw*, et *zkwm*.
- A l'initialisation : *zir*, *ziw* et *ziwm*

---

# Accueil de Plugin

Csound accueille actuellement des plugins externes au moyen de *dssi4cs* (pour les plugins LADSPA) sur Linux et *vst4cs* (pour les plugins VST) sur Windows et Mac OS X.

## DSSI et LADSPA pour Csound

*dssi4cs* permet l'utilisation des effets et des synthétiseurs plugin DSSI et LADSPA dans Csound sur Linux. Les opcodes suivants sont disponibles :

- *dssiinit* - Charge un plugin.
- *dssiactivate* - Active ou désactive un plugin si celui-ci le permet.
- *dssilist* - Liste tous les plugins disponibles trouvés dans les variables globales LADSPA\_PATH et DSSI\_PATH.
- *dssiaudio* - Traitement audio au moyen d'un Plugin.
- *dssictls* - Envoie une information de contrôle sur le port de contrôle d'un plugin.

Voir l'entrée pour *dssiinit* pour un exemple d'utilisation.



### Note

Actuellement seuls les plugins LADSPA sont supportés, mais le support de DSSI est programmé.

## VST pour Csound

*vst4cs* permet l'utilisation des effets et des synthétiseurs plugin VST dans Csound. Les opcodes suivants sont disponibles :

- *vstinit* - Charge un plugin.
- *vstaudio*, *vstaudiog* - Retourne la sortie d'un plugin.
- *vstmidiout* - Envoie des données MIDI à un plugin.
- *vstparamset*, *vstparamget* - Envoie et reçoit des données d'automatisation de et vers le plugin.
- *vstnote* - Envoie une note MIDI avec une durée définie.
- *vstinfo* - Sort les noms de Programme et de Paramètre pour un plugin.
- *vstbankload* - Charge une Banque .fxb
- *vstprogset* - Fixe un Programme dans une Banque .fxb
- *vstedit* - Ouvre l'éditeur de GUI pour le plugin, s'il est disponible.

## Crédits

Par Andrés Cabrera et Michael Gogins

Utilise du code de VSTHost par Hermann Seib et de l'objet vst~ par Thomas Grill.

VST est une marque de Steinberg Media Technologies GmbH. VST Plug-In Technology par Steinberg.



---

# OSC et Réseau

## OSC

OSC permet l'interaction entre différents processus audio, et en particulier entre Csound et d'autres moteurs de synthèse. Les opcodes suivants sont disponibles :

- *OSCinit* - Démarre un thread d'écoute OSC.
- *OSClisten* - Réçoit les messages OSC.
- *OSCsend* - Envoie un message OSC.

## Crédits

Par John ffitich avec le support et l'inspiration de la bibliothèque liblo.

## Réseau

Les opcodes suivants peuvent envoyer ou recevoir des données audio en UDP :

- *sockrecv*
- *socksend*

## Opcodes pour le Traitement à Distance

Les opcodes pour le Traitement à Distance permettent la transmission d'une partition ou d'événements MIDI à travers un réseau, pour un traitement par des instances distantes (ou une autre instance locale). Les opcodes suivants sont disponibles :

- *insglobal* - Utilisé pour implémenter un orchestre distant.
- *insremot* - Utilisé pour implémenter un orchestre distant.
- *midiglobal* - Utilisé pour implémenter un orchestre MIDI distant.
- *midiremot* - Utilisé pour implémenter un orchestre MIDI distant.
- *remoteport* - Définit le port à utiliser avec le système distant.

---

# Opcodes Mixer

La famille d'opcodes Mixer fournit un mélangeur global pour Csound. Les opcodes Mixer comprennent *MixerSend* pour envoyer (c'est-à-dire mélanger en entrée) un signal de taux-a depuis n'importe quel instrument vers un canal d'un bus de mixage, *MixerReceive* pour recevoir un signal de taux-a depuis un canal de n'importe quel bus de mixage dans un instrument, *MixerSetLevel* (taux-k) et *MixerSetLevel\_i* (taux-i) pour contrôler le niveau du signal envoyé d'une source particulière vers un bus particulier, *MixerGetLevel* pour lire (au taux-k) le niveau d'envoi d'une source particulière à un bus particulier, et *MixerClear* pour réinitialiser les bus à zéro avant la k-période suivante d'une exécution.

---

# Opcodes de Graphe de Fluence

Ces opcodes permettent d'utiliser des graphes de fluences (ou graphes de flots de données asynchrones) dans des orchestres de Csound. Les signaux s'écoulent depuis les prises de sortie (outlets) des instruments émetteurs et sont additionnés dans les prises d'entrée (inlets) des instruments récepteurs. Les signaux peuvent être de taux-k, de taux-a ou de taux-f. On peut connecter n'importe quel nombre d'outlets à n'importe quel nombre d'inlets. Lorsqu'une nouvelle instance d'un instrument est créée pendant l'exécution, les connexions déclarées sont automatiquement instanciées.

Les graphes de fluence simplifient la construction de mélangeurs complexes, de chaînes de traitement du signal, etc. Ils simplifient également la réutilisation de définitions d'instrument "plug and play" et même de sous-orchestres entiers, qui peuvent être simplement insérés (`#include`) et ainsi "branchés" dans des orchestres existants.

Noter que les inlets et les outlets sont définis dans les instruments sans référence à la manière dont ils sont connectés. Les connexions sont définies dans l'en-tête de l'orchestre. C'est cette séparation qui permet d'avoir des instruments greffons.

Les inlets doivent être nommés. Les instruments peuvent être nommés ou numérotés, mais dans tous les cas chaque instrument émetteur doit être défini dans l'orchestre avant chacun de ses récepteurs. Si les instruments sont nommés, il est plus facile de connecter les outlets et les inlets d'un orchestre de niveau plus élevé aux inlets et aux outlets d'un orchestre inclus (`#include`) de niveau moins élevé.

Les opcodes de graphe de fluence comprennent : *outleta*, pour envoyer un signal de taux-a depuis n'importe quel instrument par un port nommé. *outletk*, pour envoyer un signal de taux-k depuis n'importe quel instrument par un port nommé. *outletf*, pour envoyer un signal de taux-f depuis n'importe quel instrument par un port nommé. *inleta*, pour recevoir un signal de taux-a à travers un port nommé. *inletk*, pour recevoir un signal de taux-k à travers un port nommé. *inletf*, pour recevoir un signal de taux-f à travers un port nommé. *connect*, pour acheminer le signal depuis un outlet nommé dans un instrument émetteur vers un inlet nommé dans un instrument récepteur. *alwayson* pour activer un instrument de façon permanente depuis l'en-tête de l'orchestre, sans l'aide d'une instruction de partition, par exemple pour l'utiliser comme processeur d'effet recevant ses entrées depuis un certain nombre d'émetteurs. *figenonce* pour instancier des tables de fonction depuis des définitions d'instrument, sans l'aide d'instructions-f dans la partition ou d'opcodes *figen* dans l'en-tête de l'orchestre.

Un scénario typique d'utilisation de ces opcodes ressemble à ceci ; un ensemble d'instruments est défini, chacun dans son propre fichier d'orchestre, et chaque instrument définit des ports d'entrée, des ports de sortie et des tables de fonction en son sein. De tels instruments sont complètement autonomes. Puis un ensemble de processeurs d'effets tels qu'égaliseurs, réverbérations, compresseurs, etc, sont également définis, chacun dans son propre fichier. Enfin un orchestre maître personnalisé inclut (`#include`) les instruments et les effets à utiliser, dirige les sorties de certains instruments dans un égaliseur et les sorties d'autres effets dans un autre égaliseur, puis achemine les sorties des deux égaliseurs dans une réverbération, la sortie de la réverbération dans un compresseur et la sortie du compresseur dans un fichier de sortie stéréo.

## Exemple

Voici un exemple des opcodes de graphe de fluence. Il utilise le fichier *signalflowgraph.csd* [examples/signalflowgraph.csd].

**Exemple 7. Exemple des opcodes de graphe de fluence.**

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o madsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Michael Gogins */
; Initialize the global variables.
sr = 44100
ksmps = 100
nchnls = 2

; Connect up the instruments to create a signal flow graph.

connect "SimpleSine", "leftout", "Reverberator", "leftin"
connect "SimpleSine", "rightout", "Reverberator", "rightin"

connect "Moogy", "leftout", "Reverberator", "leftin"
connect "Moogy", "rightout", "Reverberator", "rightin"

connect "Reverberator", "leftout", "Compressor", "leftin"
connect "Reverberator", "rightout", "Compressor", "rightin"

connect "Compressor", "leftout", "Soundfile", "leftin"
connect "Compressor", "rightout", "Soundfile", "rightin"

; Turn on the "effect" units in the signal flow graph.

alwayson "Reverberator", 0.91, 12000
alwayson "Compressor"
alwayson "Soundfile"

instr SimpleSine
  ihz = cpsmidinn(p4)
  iampplitude = ampdb(p5)
  print ihz, iampplitude
  ; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
  isine ftgenonce 0, 0, 4096, 10, 1
  a1 oscili iampplitude, ihz, isine
  aenv madsr 0.05, 0.1, 0.5, 0.2
  asignal = a1 * aenv
  ; Stereo audio outlet to be routed in the orchestra header.
  outleta "leftout", asignal * 0.25
  outleta "rightout", asignal * 0.75
endin

instr Moogy
  ihz = cpsmidinn(p4)
  iampplitude = ampdb(p5)
  ; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
  isine ftgenonce 0, 0, 4096, 10, 1
  asignal vco iampplitude, ihz, 1, 0.5, isine
  kfco line 200, p3, 2000
  krez init 0.9
  asignal moogvcf asignal, kfco, krez, 100000
  ; Stereo audio outlet to be routed in the orchestra header.
  outleta "leftout", asignal * 0.75
  outleta "rightout", asignal * 0.25
endin

instr Reverberator
  ; Stereo input.
  aleftin inleta "leftin"
  arightin inleta "rightin"
  idelay = p4
  icutoff = p5
  aleftout, arightout reverbosc aleftin, arightin, idelay, icutoff
  ; Stereo output.
  outleta "leftout", aleftout
  outleta "rightout", arightout

```

```

endin

instr Compressor
; Stereo input.
aleftin inleta "leftin"
arightin inleta "rightin"
kthreshold = 25000
icomp1 = 0.5
icomp2 = 0.763
irtime = 0.1
iftime = 0.1
aleftout dam aleftin, kthreshold, icomp1, icomp2, irtime, iftime
arightout dam arightin, kthreshold, icomp1, icomp2, irtime, iftime
; Stereo output.
outleta "leftout", aleftout
outleta "rightout", arightout
endin

instr Soundfile
; Stereo input.
aleftin inleta "leftin"
arightin inleta "rightin"
outs aleftin, arightin
endin

</CsInstruments>
<CsScore>
; Not necessary to activate "effects" or create f-tables in the score!
; Overlapping notes to create new instances of instruments.
i "SimpleSine" 1 5 60 85
i "SimpleSine" 2 5 64 80
i "Moogy" 3 5 67 75
i "Moogy" 4 5 71 70
e 1
</CsScore>
</CsSoundSynthesizer>

```

---

# Opcodes Python

## Introduction

En utilisant la famille d'opcodes Python, vous pouvez interagir avec un interpréteur Python embarqué dans Csound de cinq manières :

1. Initialiser l'interpréteur Python (les opcodes *pyinit*),
2. Exécuter une instruction (les opcodes *pyrun*),
3. Exécuter un script (les opcodes *pyexec*),
4. Invoquer un objet callable et lui passer des arguments (les opcodes *pycall*),
5. Evaluer une expression (les opcodes *pyeval*), ou
6. Changer la valeur d'un objet Python, avec la possibilité de créer un nouvel objet Python (les opcodes *pyassign*) ;

et vous pouvez faire toutes ces choses :

1. Au temps-i ou au temps-k,
2. Dans l'espace de nom global de Python, ou dans un espace de nom spécifique à une instance individuelle d'un instrument Csound (contexte local ou "I"),
3. Et vous pouvez récupérer de 0 à 8 valeurs de retour d'objets appelables qui acceptent N paramètres.

...cela signifie qu'il y a beaucoup d'opcodes concernant Python. Mais tous ces opcodes partagent le même préfixe *py*, et ils ont une structure de nom régulière :

"py" + [préfixe contextuel facultatif] + [nom d'action] + [suffixe de temps-x facultatif]

## Syntaxe de l'Orchestre

Des blocs de code Python, voire des scripts entiers, peuvent être embarqués dans un orchestre Csound en utilisant les directives `{{ et }}` pour entourer le script, comme ci-dessous :

```
sr=44100
kr=4410
ksmps=10
nchnls=1
pyinit

giSinusoid ftgen 0, 0, 8192, 10, 1

pyruni {{
import random

pool = [(1 + i/10.0) ** 1.2 for i in range(100)]

def get_number_from_pool(n, p):
```

```
        if random.random() < p:
            i = int(random.random() * len(pool))
            pool[i] = n
        return random.choice(pool)
    }}

instr 1
    k1 oscil 1, 3, giSinusoid
    k2 pycall1 "get_number_from_pool", k1 + 2, p4
    printk 0.01, k2
endin
```

## Crédits

Copyright © 2002 par Maurizio Umberto Puxeddu. Tous droits réservés.

Copyright © 2004 et 2005 par Michael Gogins, pour certaines parties.

---

# Opcodes pour le traitement d'image

Voici une liste des opcodes pour lire/écrire des fichiers d'image :

- *imagecreate*
- *imagesize*
- *imagegetpixel*
- *imagesetpixel*
- *imagesave*
- *imageload*
- *imagefree*



---

# Opcodes divers

Voici une liste d'opcodes qui ne rentrent dans aucune catégorie :

- *system* - appelle un programme externe via le mécanisme d'appel du système.
- *modmatrix* - opcode matrice de modulation avec optimisation pour les matrices creuses.

---

## **Partie III. Référence**

---

---

## Table des matières

Opcodes et Opérateurs de l'Orchestre .....	203
!= .....	204
#define .....	206
#include .....	210
#undef .....	212
#ifdef .....	213
#ifndef .....	215
\$NOM .....	216
% .....	219
&& .....	221
> .....	222
>= .....	224
< .....	226
<= .....	228
* .....	230
+ .....	232
- .....	234
/ .....	236
= .....	238
== .....	240
^ .....	242
.....	244
Odbfs .....	245
<< .....	248
>> .....	250
& .....	251
.....	253
¬ .....	254
# .....	255
a .....	256
abetarand .....	258
abexprnd .....	259
abs .....	260
acauchy .....	262
active .....	263
adsr .....	267
adsyn .....	270
adsynt .....	272
adsynt2 .....	275
aexprand .....	277
aftouch .....	278
agauss .....	280
agogobel .....	281
alinrand .....	282
alpass .....	283
alwayson .....	286
ampdb .....	287
ampdbfs .....	289
ampmidi .....	291
apcauchy .....	293
apoisson .....	294
apow .....	295
areson .....	296

aresonk .....	298
atone .....	299
atonek .....	301
atonex .....	302
atirand .....	303
ATSadd .....	304
ATSaddnz .....	306
ATSbufread .....	308
ATScross .....	310
ATSinfor .....	312
ATSinterpread .....	314
ATSread .....	315
ATSreadnz .....	317
ATSpartialtap .....	319
ATSinnoi .....	320
aunirand .....	322
aweibull .....	323
babo .....	324
balance .....	328
bamboo .....	330
barmodel .....	332
bbcutm .....	334
bbcuts .....	339
betarand .....	341
bexprnd .....	343
bformenc .....	345
bformenc1 .....	347
bformdec .....	349
bformdec1 .....	351
binit .....	353
biquad .....	354
biquada .....	358
birnd .....	359
bqrez .....	361
butbp .....	363
butbr .....	364
buthp .....	365
butlp .....	366
butterbp .....	367
butterbr .....	369
butterhp .....	371
butterlp .....	373
button .....	375
buzz .....	376
cabasa .....	378
cauchy .....	380
ceil .....	382
cent .....	383
cggoto .....	385
chanctrl .....	387
changed .....	388
chani .....	390
chano .....	391
chebyshevpoly .....	392
checkbox .....	395
chn .....	397
chnclear .....	399
chnexport .....	400

chnget .....	402
chnmix .....	404
chnparams .....	405
chnrecv .....	406
chnsend .....	408
chnset .....	410
chuap .....	412
cigoto .....	416
ckgoto .....	418
clear .....	420
clfilt .....	421
clip .....	424
clock .....	427
clockoff .....	428
clockon .....	429
cngoto .....	430
comb .....	432
compress .....	435
connect .....	437
control .....	438
convle .....	439
convolve .....	440
cos .....	443
cosh .....	445
cosinv .....	447
cps2pch .....	449
cpsmidi .....	453
cpsmidib .....	455
cpsmidinn .....	457
cpsoct .....	460
cpspch .....	463
cpstmid .....	466
cpstun .....	469
cpstuni .....	472
cpsexpch .....	475
cpuprc .....	479
cross2 .....	481
crossfm .....	483
crunch .....	486
ctrl14 .....	488
ctrl21 .....	490
ctrl7 .....	492
ctrlinit .....	494
cusernd .....	495
dam .....	497
date .....	501
dates .....	503
db .....	505
dbamp .....	507
dbfsamp .....	509
dcblock .....	511
dcblock2 .....	513
dconv .....	514
delay .....	516
delay1 .....	518
delayk .....	519
delayr .....	521
delayw .....	522

deltap .....	524
deltap3 .....	526
deltapi .....	528
deltapn .....	530
deltapx .....	532
deltapxw .....	534
denorm .....	536
diff .....	537
diskgrain .....	539
diskin .....	542
diskin2 .....	545
dispfft .....	548
display .....	550
distort .....	552
distort1 .....	554
divz .....	556
doppler .....	558
downsamp .....	560
dripwater .....	562
dssiactivate .....	564
dssiaudio .....	565
dssictls .....	566
dssiinit .....	567
dssilist .....	569
dumpk .....	570
dumpk2 .....	572
dumpk3 .....	574
dumpk4 .....	576
dusernd .....	578
else .....	580
elseif .....	581
endif .....	582
endin .....	583
endop .....	585
envlpx .....	586
envlpxr .....	589
ephasor .....	591
eqfil .....	592
event .....	594
event_i .....	597
exitnow .....	599
exp .....	600
expcurve .....	602
expon .....	604
expvand .....	606
expseg .....	608
expsega .....	610
expsegr .....	612
ficlose .....	615
filebit .....	617
filelen .....	619
filenchnls .....	621
filepeak .....	623
filesr .....	625
filter2 .....	627
fin .....	629
fini .....	631
fink .....	633

fiopen .....	635
flanger .....	637
flashtxt .....	639
FLbox .....	641
FLbutBank .....	646
FLbutton .....	649
FLcloseButton .....	654
FLcolor .....	657
FLcolor2 .....	659
FLcount .....	660
FLexecButton .....	663
FLgetsnap .....	666
FLgroup .....	668
FLgroupEnd .....	670
FLgroupEnd .....	671
FLhide .....	672
FLhvsBox .....	673
FLhvsBoxSetValue .....	674
FLjoy .....	675
FLkeyIn .....	678
FLknob .....	680
FLlabel .....	685
FLloadsnap .....	687
FLmouse .....	688
flooper .....	690
flooper2 .....	692
floor .....	694
FLpack .....	695
FLpackEnd .....	698
FLpack_end .....	699
FLpanel .....	700
FLpanelEnd .....	703
FLpanel_end .....	704
FLprintk .....	705
FLprintk2 .....	706
FLroller .....	707
FLrun .....	710
FLsavesnap .....	711
FLscroll .....	716
FLscrollEnd .....	719
FLscroll_end .....	720
FLsetAlign .....	721
FLsetBox .....	722
FLsetColor .....	724
FLsetColor2 .....	726
FLsetFont .....	727
FLsetPosition .....	729
FLsetSize .....	730
FLsetsnap .....	731
FLsetSnapGroup .....	733
FLsetText .....	734
FLsetTextColor .....	736
FLsetTextSize .....	737
FLsetTextType .....	738
FLsetVal_i .....	741
FLsetVal .....	742
FLshow .....	743
FLslidBnk .....	744

FLslidBnk2 .....	748
FLslidBnkGetHandle .....	751
FLslidBnkSet .....	752
FLslidBnkSetk .....	753
FLslidBnk2Set .....	755
FLslidBnk2Setk .....	756
FLslider .....	759
FLtabs .....	765
FLtabsEnd .....	770
FLtabs_end .....	771
FLtext .....	772
FLupdate .....	775
fluidAllOut .....	776
fluidCCi .....	779
fluidCCk .....	780
fluidControl .....	781
fluidEngine .....	783
fluidLoad .....	787
fluidNote .....	789
fluidOut .....	791
fluidProgramSelect .....	793
fluidSetInterpMethod .....	795
FLvalue .....	796
FLvkeybd .....	798
FLvslidBnk .....	799
FLvslidBnk2 .....	803
FLxyin .....	806
fmb3 .....	809
fmbell .....	812
fmmetal .....	815
fmpercfl .....	818
fmrhode .....	821
fmvoice .....	824
fmwurlie .....	826
fof .....	829
fof2 .....	832
fofilter .....	838
fog .....	840
fold .....	842
follow .....	844
follow2 .....	846
foscil .....	848
foscili .....	850
fout .....	852
fouti .....	857
foutir .....	859
foutk .....	861
fprintks .....	863
fprints .....	869
frac .....	871
freeverb .....	873
ftchnls .....	875
ftconv .....	877
ftfree .....	880
ftgen .....	881
ftgenonce .....	884
ftgentmp .....	886
ftlen .....	888



ftload .....	890
ftloadk .....	891
ftlptim .....	892
ftmorf .....	894
ftsave .....	896
ftsavek .....	898
ftsr .....	899
gain .....	901
gainslider .....	903
gauss .....	905
gbuzz .....	907
getcfcg .....	910
gogobel .....	911
goto .....	913
grain .....	915
grain2 .....	917
grain3 .....	921
granule .....	926
guiro .....	930
harmon .....	932
harmon2 .....	935
hilbert .....	937
hrtfer .....	941
hrtfmove .....	943
hrtfmove2 .....	946
hrtfstat .....	949
hsboscil .....	952
hvs1 .....	955
hvs2 .....	959
hvs3 .....	965
i .....	968
ibetarand .....	969
ibexprnd .....	970
icauchy .....	971
ictrl14 .....	972
ictrl21 .....	973
ictrl7 .....	974
iexprand .....	975
if .....	976
igauss .....	981
igoto .....	982
ihold .....	984
ilinrand .....	986
imagecreate .....	987
imagefree .....	989
imagegetpixel .....	991
imageload .....	993
imagesave .....	995
imagesetpixel .....	997
imagesize .....	999
imidic14 .....	1001
imidic21 .....	1002
imidic7 .....	1003
in .....	1004
in32 .....	1005
inch .....	1006
inh .....	1007
init .....	1008

initc14 .....	1009
initc21 .....	1010
initc7 .....	1011
inleta .....	1012
inletk .....	1013
inletf .....	1014
ino .....	1015
inq .....	1016
inrg .....	1017
ins .....	1018
insremot .....	1019
insglobal .....	1021
instimek .....	1022
instimes .....	1023
instr .....	1024
int .....	1027
integ .....	1029
interp .....	1031
invalue .....	1034
inx .....	1035
inz .....	1036
ioff .....	1037
ion .....	1038
iondur .....	1039
iondur2 .....	1040
ioutat .....	1041
ioutc .....	1042
ioutc14 .....	1043
ioutpat .....	1044
ioutpb .....	1045
ioutpc .....	1046
ipcauchy .....	1047
ipoisson .....	1048
ipow .....	1049
is16b14 .....	1050
is32b14 .....	1051
islider16 .....	1052
islider32 .....	1053
islider64 .....	1054
islider8 .....	1055
itablecopy .....	1056
itablegpw .....	1057
itablemix .....	1058
itablew .....	1059
itrirand .....	1060
iunirand .....	1061
iweibull .....	1062
jacktransport .....	1063
jitter .....	1065
jitter2 .....	1067
jspline .....	1069
k .....	1070
kbetarand .....	1071
kbexprnd .....	1072
kcauchy .....	1073
kdump .....	1074
kdump2 .....	1075
kdump3 .....	1076

kdump4 .....	1077
kexprand .....	1078
kfilter2 .....	1079
kgauss .....	1080
kgoto .....	1081
klinrand .....	1083
kon .....	1084
koutat .....	1085
koutc .....	1086
koutc14 .....	1087
koutpat .....	1088
koutpb .....	1089
koutpc .....	1090
kpcauchy .....	1091
kpoisson .....	1092
kpow .....	1093
kr .....	1094
kread .....	1095
kread2 .....	1096
kread3 .....	1097
kread4 .....	1098
ksmps .....	1099
ktableseg .....	1100
ktirand .....	1101
kunirand .....	1102
kweibull .....	1103
lfo .....	1104
limit .....	1106
line .....	1107
linen .....	1109
linenr .....	1111
lineto .....	1112
linrand .....	1113
linseg .....	1115
linsegr .....	1118
locsend .....	1121
locsig .....	1123
log .....	1126
log10 .....	1128
logbtwo .....	1130
logcurve .....	1132
loop_ge .....	1134
loop_gt .....	1135
loop_le .....	1136
loop_lt .....	1137
loopseg .....	1138
loopsegg .....	1140
looptseg .....	1142
loopxseg .....	1144
lorenz .....	1146
lorisread .....	1149
lorismorph .....	1151
lorisplay .....	1152
loscil .....	1153
loscil3 .....	1156
loscilx .....	1159
lowpass2 .....	1160
lowres .....	1162

lowresx .....	1164
lpf18 .....	1166
lpfreson .....	1168
lphasor .....	1169
lpinterp .....	1171
lposcil .....	1172
lposcil3 .....	1173
lposcila .....	1174
lposcilsa .....	1175
lposcilsa2 .....	1176
lpread .....	1177
lpreson .....	1179
lpshold .....	1180
lpsholdp .....	1182
lpslot .....	1183
mac .....	1185
maca .....	1186
madsr .....	1187
mandel .....	1190
mandol .....	1191
marimba .....	1193
massign .....	1196
max .....	1198
maxabs .....	1199
maxabsaccum .....	1200
maxaccum .....	1201
maxalloc .....	1202
max_k .....	1204
mclock .....	1205
mdelay .....	1206
metro .....	1208
midic14 .....	1210
midic21 .....	1212
midic7 .....	1214
midichannelaftertouch .....	1216
midichn .....	1218
midicontrolchange .....	1221
midictrl .....	1223
mididefault .....	1224
midiin .....	1225
midinoteoff .....	1228
midinoteoncps .....	1230
midinoteonkey .....	1232
midinoteonoct .....	1234
midinoteonpch .....	1236
midion .....	1238
midion2 .....	1241
midiout .....	1242
midipitchbend .....	1244
midipolyaftertouch .....	1246
midiprogramchange .....	1248
miditempo .....	1249
midremot .....	1250
midglobal .....	1253
min .....	1254
minabs .....	1255
minabsaccum .....	1256
minaccum .....	1257

mirror .....	1258
MixerSetLevel .....	1259
MixerSetLevel_i .....	1261
MixerGetLevel .....	1262
MixerSend .....	1263
MixerReceive .....	1264
MixerClear .....	1266
mode .....	1267
modmatrix .....	1270
monitor .....	1275
moog .....	1276
moogladder .....	1278
moogvcf .....	1280
moogvcf2 .....	1282
moscil .....	1284
mp3in .....	1286
mpulse .....	1287
mrtmsg .....	1289
multitap .....	1290
mute .....	1291
mxadsr .....	1293
nchnls .....	1295
nestedap .....	1296
nlfilt .....	1299
noise .....	1301
noteoff .....	1304
noteon .....	1305
noteondur .....	1306
noteondur2 .....	1308
notnum .....	1310
nreverb .....	1312
nrpn .....	1315
nsamp .....	1316
nstrnum .....	1318
ntrpol .....	1319
octave .....	1320
octcps .....	1322
octmidi .....	1325
octmidib .....	1327
octmidinn .....	1329
octpch .....	1332
opcode .....	1335
OSCsend .....	1340
OSCinit .....	1342
OSClisten .....	1343
oscbnk .....	1347
oscil .....	1352
oscil1 .....	1354
oscil1i .....	1355
oscil3 .....	1356
oscili .....	1358
oscilikt .....	1360
osciliktp .....	1362
oscilikts .....	1364
osciln .....	1366
oscils .....	1367
oscilx .....	1369
out .....	1370

out32 .....	1371
outc .....	1372
outch .....	1373
outh .....	1374
outiat .....	1375
outic .....	1376
outic14 .....	1377
outipat .....	1379
outipb .....	1380
outipc .....	1381
outkat .....	1382
outkc .....	1383
outkc14 .....	1384
outkpat .....	1385
outkpb .....	1386
outkpc .....	1387
outleta .....	1390
outletk .....	1391
outletf .....	1392
outo .....	1393
outq .....	1394
outq1 .....	1395
outq2 .....	1396
outq3 .....	1397
outq4 .....	1398
outrg .....	1399
outs .....	1400
outs1 .....	1401
outs2 .....	1402
outvalue .....	1403
outx .....	1404
outz .....	1405
p .....	1406
p5gconnect .....	1408
p5gdata .....	1410
pan .....	1412
pan2 .....	1414
pareq .....	1415
partials .....	1418
partikkel .....	1420
partikkelsync .....	1429
passign .....	1430
pcauchy .....	1432
pchbend .....	1434
pchmidi .....	1436
pchmidib .....	1438
pchmidinn .....	1440
pchoct .....	1443
pconvolve .....	1446
pcount .....	1449
pdclip .....	1451
pdhalf .....	1454
pdhalfy .....	1457
peak .....	1460
peakk .....	1462
pgmassign .....	1463
phaser1 .....	1467
phaser2 .....	1470

phasor .....	1474
phasorbnk .....	1476
pindex .....	1478
pinkish .....	1480
pitch .....	1483
pitchamdf .....	1486
planet .....	1489
pluck .....	1491
poisson .....	1494
polyaft .....	1498
polynomial .....	1500
pop .....	1502
pop_f .....	1503
port .....	1504
portk .....	1505
poscil .....	1507
poscil3 .....	1510
pow .....	1513
powershape .....	1515
powoftwo .....	1517
prealloc .....	1519
prepiano .....	1521
print .....	1524
printf .....	1526
printk .....	1527
printk2 .....	1529
printks .....	1531
prints .....	1534
product .....	1536
pset .....	1537
ptrack .....	1538
puts .....	1540
push .....	1541
push_f .....	1542
pvadd .....	1543
pvbufread .....	1546
pvcross .....	1548
pvinterp .....	1550
pvoc .....	1552
pvread .....	1554
pvsadsyn .....	1556
pvsanal .....	1558
pvsarp .....	1561
pvsbandp .....	1564
pvsbandr .....	1566
pvsbin .....	1568
pvsblur .....	1570
pvsbuffer .....	1572
pvsbufread .....	1573
pvscale .....	1576
pvscent .....	1578
pvsccross .....	1580
pvsdemix .....	1582
pvsdiskin .....	1584
pvsdisp .....	1585
pvsfilter .....	1587
pvsfread .....	1590
pvsfreeze .....	1591

pvsftr .....	1593
pvsftw .....	1595
pvsfwrite .....	1597
pvshift .....	1599
pvsifd .....	1601
pvsinfo .....	1603
pvsinit .....	1604
pvsin .....	1605
pvsmaska .....	1606
pvsmix .....	1608
pvsmorph .....	1609
pvsMOOTH .....	1612
pvsout .....	1614
pvsosc .....	1615
pvspitch .....	1618
pvstencil .....	1621
pvsvoc .....	1623
pvsynth .....	1625
pyassign Opcodes .....	1627
pycall Opcodes .....	1628
pyeval Opcodes .....	1632
pyexec Opcodes .....	1633
pyinit Opcodes .....	1636
pyrun Opcodes .....	1637
rand .....	1639
randh .....	1641
randi .....	1643
random .....	1645
randomh .....	1647
randomi .....	1649
rbjeq .....	1651
readclock .....	1654
readk .....	1656
readk2 .....	1658
readk3 .....	1660
readk4 .....	1662
reinit .....	1664
release .....	1666
remoteport .....	1667
remove .....	1668
repluck .....	1669
reson .....	1671
resonk .....	1673
resonr .....	1674
resonx .....	1677
resonxk .....	1679
resony .....	1680
resonz .....	1682
resyn .....	1684
reverb .....	1686
reverb2 .....	1688
reverb3 .....	1689
rewindscore .....	1691
rezzy .....	1692
rigoto .....	1694
rireturn .....	1695
rms .....	1697
rnd .....	1699



rnd31 .....	1701
round .....	1706
rspline .....	1707
rtclock .....	1708
s16b14 .....	1710
s32b14 .....	1712
scale .....	1714
samphold .....	1716
sandpaper .....	1717
scanhammer .....	1719
scans .....	1720
scantable .....	1722
scanu .....	1724
scoreline .....	1726
scoreline_i .....	1728
schedkwhen .....	1730
schedkwhennamed .....	1733
schedule .....	1735
schedwhen .....	1738
seed .....	1741
sekere .....	1742
semitone .....	1744
sense .....	1746
sensekey .....	1747
seqtime .....	1751
seqtime2 .....	1754
setctrl .....	1756
setksmps .....	1758
setscorepos .....	1760
sfilist .....	1761
sfinstr .....	1762
sfinstr3 .....	1764
sfinstr3m .....	1766
sfinstrm .....	1768
sfload .....	1770
sflooper .....	1771
sfpassign .....	1773
sfplay .....	1775
sfplay3 .....	1777
sfplay3m .....	1779
sfplaym .....	1781
sfplist .....	1783
sfpreset .....	1784
shaker .....	1786
sin .....	1788
sinh .....	1790
sininv .....	1792
sinsyn .....	1794
sleighbells .....	1796
slider16 .....	1798
slider16f .....	1800
slider32 .....	1802
slider32f .....	1804
slider64 .....	1806
slider64f .....	1808
slider8 .....	1810
slider8f .....	1812
slider16table .....	1814

slider16tablef .....	1816
slider32table .....	1818
slider32tablef .....	1820
slider64table .....	1822
slider64tablef .....	1824
slider8table .....	1826
slider8tablef .....	1828
sliderKawai .....	1830
sndload .....	1831
sndloop .....	1833
sndwarp .....	1835
sndwarpst .....	1839
socksend .....	1842
sockrecv .....	1844
soundin .....	1846
soundout .....	1849
soundouts .....	1851
space .....	1853
spat3d .....	1857
spat3di .....	1865
spat3dt .....	1869
spdist .....	1873
specaddm .....	1877
specdiff .....	1878
specdisp .....	1879
specfilt .....	1880
spechist .....	1881
specptrk .....	1882
specscal .....	1884
specsum .....	1885
spectrum .....	1886
splitrig .....	1888
spsend .....	1890
sprintf .....	1893
sprintfk .....	1894
sqrt .....	1896
sr .....	1898
stack .....	1899
statevar .....	1900
stix .....	1902
strchar .....	1904
strchark .....	1905
strcpy .....	1906
strcpyk .....	1907
strcat .....	1908
strcatk .....	1909
strcmp .....	1910
strcmpk .....	1911
streson .....	1912
strget .....	1914
strindex .....	1915
strindexk .....	1916
strlen .....	1917
strlenk .....	1918
strlower .....	1919
strlowerk .....	1920
strrindex .....	1921
strrindexk .....	1922

strset .....	1923
strsub .....	1925
strsubk .....	1927
strtod .....	1928
strtodk .....	1929
strtol .....	1930
strtolk .....	1931
strupper .....	1932
strupperk .....	1933
subinstr .....	1934
subinstrinit .....	1937
sum .....	1938
svfilter .....	1939
syncgrain .....	1942
syncloop .....	1944
syncphasor .....	1946
system .....	1950
tb .....	1952
tab .....	1955
tabrec .....	1957
table .....	1958
table3 .....	1960
tablecopy .....	1961
tablegpw .....	1962
tablei .....	1963
tableicopy .....	1964
tableigpw .....	1965
tableikt .....	1966
tableimix .....	1968
tableiw .....	1970
tablekt .....	1972
tablemix .....	1974
tableng .....	1976
tablera .....	1978
tableseg .....	1981
tablew .....	1982
tablewa .....	1985
tablewkt .....	1988
tablexkt .....	1991
tablexseg .....	1994
tabmorph .....	1995
tabmorpha .....	1997
tabmorphak .....	1999
tabmorphi .....	2001
tabplay .....	2003
tabsum .....	2004
tambourine .....	2005
tan .....	2007
tanh .....	2009
taninv .....	2011
taninv2 .....	2013
tbvcf .....	2015
tempest .....	2018
tempo .....	2021
tempoval .....	2023
tigoto .....	2025
timedseq .....	2026
timeinstk .....	2028

timeinsts .....	2030
timek .....	2032
times .....	2034
timeout .....	2036
tival .....	2037
tlineto .....	2038
tone .....	2039
tonek .....	2040
tonex .....	2041
trandom .....	2042
tradsyn .....	2043
transeg .....	2045
transegr .....	2047
trcross .....	2049
trfilter .....	2051
trhighest .....	2053
trigger .....	2054
trigseq .....	2056
trirand .....	2058
trlowest .....	2060
trmix .....	2061
trscale .....	2062
trshift .....	2063
trsplitt .....	2064
turnoff .....	2066
turnoff2 .....	2068
turnon .....	2069
unirand .....	2070
upsamp .....	2072
urd .....	2073
vadd .....	2074
vadd_i .....	2077
vaddv .....	2079
vaddv_i .....	2082
vaget .....	2084
valpass .....	2086
vaset .....	2087
vbap16 .....	2089
vbap16move .....	2091
vbap4 .....	2093
vbap4move .....	2095
vbap8 .....	2097
vbap8move .....	2099
vbaplsinit .....	2102
vbapz .....	2104
vbapzmove .....	2106
vcella .....	2108
vco .....	2111
vco2 .....	2114
vco2ft .....	2118
vco2ift .....	2120
vco2init .....	2122
vcomb .....	2125
vcopy .....	2128
vcopy_i .....	2131
vdelay .....	2133
vdelay3 .....	2135
vdelayx .....	2137

vdelayxq .....	2139
vdelayxs .....	2141
vdelayxw .....	2143
vdelayxwq .....	2145
vdelayxws .....	2147
vdivv .....	2149
vdivv_i .....	2152
vdelayk .....	2154
vecdelay .....	2155
veloc .....	2156
vexp .....	2158
vexp_i .....	2161
vexpseg .....	2163
vexpv .....	2165
vexpv_i .....	2168
vibes .....	2170
vibr .....	2172
vibrato .....	2174
vincr .....	2177
vlimit .....	2178
vlinseg .....	2179
vlowres .....	2181
vmap .....	2183
vmirror .....	2185
vmult .....	2186
vmult_i .....	2190
vmultv .....	2192
vmultv_i .....	2195
voice .....	2197
vosim .....	2200
vphaseseg .....	2205
vport .....	2207
vpow .....	2208
vpow_i .....	2211
vpowv .....	2213
vpowv_i .....	2216
vpvoc .....	2218
vrandh .....	2220
vrandi .....	2223
vstaudio, vstaudiog .....	2226
vstbankload .....	2228
vstedit .....	2229
vstinit .....	2231
vstinfo .....	2233
vstmidiout .....	2235
vstnote .....	2237
vstparamset, vstparamget .....	2239
vstprogset .....	2241
vsubv .....	2242
vsubv_i .....	2245
vtable1k .....	2247
vtablei .....	2249
vtablek .....	2251
vtablea .....	2253
vtablewi .....	2255
vtablewk .....	2256
vtablewa .....	2258
vtabi .....	2260

vtabk .....	2262
vtaba .....	2264
vtabwi .....	2266
vtabwk .....	2267
vtabwa .....	2268
vwrap .....	2269
waveset .....	2270
weibull .....	2272
wgbow .....	2274
wgbowedbar .....	2276
wgbrass .....	2278
wgclar .....	2280
wgflute .....	2282
wgpluck .....	2284
wgpluck2 .....	2287
wguide1 .....	2289
wguide2 .....	2291
wiiconnect .....	2294
wiidata .....	2296
wiirange .....	2298
wiisend .....	2299
wrap .....	2300
wterrain .....	2301
xadsr .....	2303
xin .....	2305
xout .....	2307
xscanmap .....	2309
xscansmap .....	2310
xscans .....	2311
xscanu .....	2313
xtratim .....	2316
xyin .....	2319
zacl .....	2321
zakinit .....	2323
zamod .....	2326
zar .....	2328
zarg .....	2330
zaw .....	2332
zawm .....	2334
zfilter2 .....	2337
zir .....	2339
ziw .....	2341
ziwm .....	2343
zkcl .....	2345
zkmod .....	2347
zkr .....	2349
zkw .....	2351
zkwm .....	2353
Instructions de Partition et Routines GEN .....	2356
Instructions de Partition .....	2356
Instruction a (ou Instruction Avancer) .....	2357
Instruction b .....	2358
Instruction e .....	2359
Instruction f (ou Instruction de Table de Fonction) .....	2360
Instruction i (Instruction d'Instrument ou de Note) .....	2363
Instruction m (Instruction de Marquage) .....	2367
Instruction n .....	2368
Instruction q .....	2370

Instruction r (Instruction Répéter) .....	2371
Instruction s .....	2373
Instruction t (Instruction de Tempo) .....	2374
Instruction v .....	2375
Instruction x .....	2377
Instruction { .....	2378
Instruction } .....	2381
Routines GEN .....	2381
GEN01 .....	2385
GEN02 .....	2388
GEN03 .....	2390
GEN04 .....	2392
GEN05 .....	2394
GEN06 .....	2396
GEN07 .....	2398
GEN08 .....	2400
GEN09 .....	2402
GEN10 .....	2405
GEN11 .....	2407
GEN12 .....	2409
GEN13 .....	2411
GEN14 .....	2414
GEN15 .....	2417
GEN16 .....	2418
GEN17 .....	2421
GEN18 .....	2422
GEN19 .....	2423
GEN20 .....	2425
GEN21 .....	2427
GEN22 .....	2429
GEN23 .....	2430
GEN24 .....	2431
GEN25 .....	2432
GEN27 .....	2433
GEN28 .....	2434
GEN30 .....	2436
GEN31 .....	2437
GEN32 .....	2438
GEN33 .....	2440
GEN34 .....	2442
GEN40 .....	2444
GEN41 .....	2445
GEN42 .....	2446
GEN43 .....	2447
GEN49 .....	2448
GEN51 .....	2450
GEN52 .....	2452
Les Programmes Utilitaires .....	2453
Répertoires. ....	2453
Formats des Fichiers Son. ....	2453
Génération d'un Fichier d'Analyse (ATSA, CVANAL, HETRO, LPANAL, PVANAL) .....	2454
Requêtes sur un Fichier (SNDINFO) .....	2465
Conversion de Fichier (, HET_EXPORT, HET_IMPORT, PVLOOK, PV_EXPORT, PV_IMPORT, SDIF2AD, SRCONV) .....	2467
Autres Utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MA-KECSD, MIXER, SCALE) .....	2482
Cscore .....	2495

Evénements, Listes et Opérations .....	2495
Ecrire un Programme de Contrôle Cscore .....	2498
Compiler un Programme Cscore .....	2503
Exemples Plus Avancés .....	2506
Etendre Csound .....	2508
Ajouter des Générateurs Unitaires .....	2508
Créer un Générateur Unitaire Intégré .....	2508
Ajouter un Générateur Unitaire comme Plugin .....	2512
Référence de OENTRY .....	2512



---

# Opcodes et Opérateurs de l'Orchestre

# !=

!= — Détermine si une valeur n'est pas égale à l'autre.

## Description

Détermine si une valeur n'est pas égale à l'autre.

## Syntaxe

```
( a != b ? v1 : v2 )
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* n'est pas égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur !=. Il utilise le fichier *notequal.csd* [examples/notequal.csd].

### Exemple 8. Exemple de l'opérateur !=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o notequal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```

instr 1
; Get the 4th p-field from the score.
k1 = p4

; Is it not equal to 3? (1 = true, 0 = false)
k2 = (p4 != 3 ? 1 : 0)

; Print the values of k1 and k2.
printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin

</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

k1 = 2.000000, k2 = 1.000000
k1 = 3.000000, k2 = 0.000000
k1 = 4.000000, k2 = 1.000000

```

## Voir Aussi

`==`, `>=`, `>`, `<=`, `<`

## Crédits

Exemple écrit par Kevin Conder.

# #define

#define — Définit une macro.

## Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et \$ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

*#define NOM* -- définit une macro simple. Le nom de la macro doit commencer par une lettre et peut comprendre n'importe quelle combinaison de lettres et de chiffres. La casse est significative. Cette forme est limitée dans le sens que les noms de variable sont fixes. On peut obtenir plus de flexibilité en utilisant une macro avec arguments, décrite ci-dessous.

*#define NOM(a' b' c')* -- définit une macro avec arguments. On peut l'utiliser dans des situations plus complexes. Le nom de la macro doit commencer par une lettre et peut comprendre n'importe quelle combinaison de lettres et de chiffres. Dans le texte de substitution, les arguments sont appelés sous la forme : \$A. En fait, l'implémentation définit les arguments comme des macros simples. Il peut y avoir jusqu'à 5 arguments, et les noms sont une combinaison quelconque de lettres. Souvenez-vous que la casse est significative dans les noms de macro.

## Syntax

```
#define NOM # texte de substitution #
```

```
#define NOM(a' b' c') # texte de substitution #
```

## Initialisation

*# texte de substitution #* -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est entouré par des caractères #, ce qui garantit qu'aucun caractère supplémentaire ne sera capturé par inadvertance.

## Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

## Exemples

Voici un exemple simple de définition de macro. Il utilise le fichier *define.csd* [examples/define.csd].

### Exemple 9. Exemple simple de définition de macro.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o define.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the macros.
#define VOLUME #5000#
#define FREQ #440#
#define TABLE #1#

; Instrument #1
instr 1
; Use the macros.
; This will be expanded to "a1 oscil 5000, 440, 1".
a1 oscil $VOLUME, $FREQ, $TABLE

; Send it to the output.
out a1
endin

</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie présentera des lignes comme celles-ci :

```
Macro definition for VOLUME
Macro definition for CPS
Macro definition for TABLE
```

Voici un exemple simple de définition de macro avec arguments. Il utilise le fichier *define\_args.csd* [examples/define\_args.csd].

## Exemple 10. Exemple simple de définition de macro avec arguments.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o define_args.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```

kr = 4410
ksmps = 10
nchnls = 1

; Define the oscillator macro.
#define OSCMACRO(VOLUME'FREQ'TABLE) #oscil $VOLUME, $FREQ, $TABLE#

; Instrument #1
instr 1
; Use the oscillator macro.
; This will be expanded to "a1 oscil 5000, 440, 1".
a1 $OSCMACRO(5000'440'1)

; Send it to the output.
out a1
endin

</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

La sortie présentera des lignes comme celle-ci :

Macro definition for OSCMACRO

## Macros Prédéfinies de Constantes Mathématiques

A partir de Csound 5.04 des Macros de Constantes Mathématiques sont prédéfinies. Les valeurs définies sont celles que l'on trouve dans le fichier d'en-tête C math.h, et elles sont automatiquement définies au démarrage de Csound et disponibles pour une utilisation dans les orchestres.

Macro	Valeur	Equivalent à
\$M_E	2.7182818284590452354	e
\$M_LOG2E	1.4426950408889634074	log <sub>2</sub> (e)
\$M_LOG10E	0.43429448190325182765	log <sub>10</sub> (e)
\$M_LN2	0.69314718055994530942	log <sub>e</sub> (2)
\$M_LN10	2.30258509299404568402	log <sub>e</sub> (10)
\$M_PI	3.14159265358979323846	pi
\$M_PI_2	1.57079632679489661923	pi/2
\$M_PI_4	0.78539816339744830962	pi/4
\$M_1_PI	0.31830988618379067154	1/pi
\$M_2_PI	0.63661977236758134308	2/pi
\$M_2_SQRTPI	1.12837916709551257390	2/sqrt(pi)
\$M_SQRT2	1.41421356237309504880	sqrt(2)
\$M_SQRT1_2	0.70710678118654752440	1/sqrt(2)

## Voir Aussi

*\$NOM, #undef*

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.48 de Csound

# #include

#include — Inclut un fichier externe pour traitement.

## Description

Inclut un fichier externe pour traitement.

## Syntaxe

```
#include "nomfichier"
```

## Exécution

Il est parfois commode d'organiser un orchestre sur plusieurs fichiers, par exemple avec chaque instrument dans un fichier séparé. Ce style est supporté par la fonctionnalité *#include* qui fait partie du système de macros. Une ligne contenant le texte

```
#include "nomfichier"
```

où le caractère " peut être remplacé par n'importe quel caractère approprié. Pour la plupart des utilisations le symbole de l'apostrophe double sera probablement le plus commode. Le nom de fichier peut inclure un chemin complet.

L'entrée est prise à partir du fichier nommé jusqu'à son terme, puis revient à la source précédente. *Note* : les versions de Csound antérieures à la 4.19 limitaient à 20 la profondeur des fichiers inclus et des macros.

Il est également suggéré d'utiliser *#include* pour définir un ensemble de macros qui font partie du style du compositeur.

A la limite, on pourrait définir chaque instrument comme une macro, avec un numéro d'instrument en paramètre. On pourrait alors construire un orchestre entier à partir d'un certain nombre d'instructions *#include* suivies par des appels de macro.

```
#include "clarinet"  
#include "flute"  
#include "bassoon"  
$CLARINET(1)  
$FLUTE(2)  
$BASSOON(3)
```

Il faut insister sur le fait que ces changements ont lieu au niveau littéral et n'ont donc aucune incidence sémantique.

## Exemples

Voici un exemple de l'opcode include. Il utilise les fichiers *include.csd* [examples/include.csd], et *table1.inc* [examples/table1.inc].



## Exemple 11. Exemple de l'opcode include.

```
/* table1.inc */
; Table #1, a sine wave.
f 1 0 16384 10 1
/* table1.inc */
```

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o include.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  al oscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Include the file for Table #1.
#include "table1.inc"

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.48 de Csound

# #undef

#undef — Annule la définition d'une macro.

## Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et \$ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

*#undef NOM* -- annule la définition d'un nom de macro. Si une macro n'est plus nécessaire, on peut annuler sa définition avec *#undef NOM*.

## Syntaxe

**#undef** NOM

## Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

## Voir Aussi

*#define*, *\$NOM*

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Nouveau dans la version 3.48 de Csound

# #ifdef

#ifdef — Lecture de code conditionnelle.

## Description

Si une macro est définie alors *#ifdef* peut incorporer du texte dans un orchestre jusqu'au prochain *#end*. C'est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

## Syntaxe

```
#ifdef NOM  
  
....  
  
#else  
  
....  
  
#end
```

## Exécution

Noter que l'on peut imbriquer les *#ifdef*, comme dans le langage du préprocesseur C.

## Exemples

Voici un exemple simple de cette insertion conditionnelle.

### Exemple 12. Exemple simple de la forme *#ifdef*.

```
#define debug  
  instr 1  
#ifdef debug  
  print "calling oscil"  
#end  
al  oscil 32000,440,1  
out al  
endin
```

## Voir Aussi

*\$NOM*, *#define*, *#ifndef*.

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.

Bath, UK  
Avril 2005

Nouveau dans Csound5 (et 4.23f13)

# #ifndef

#ifndef — Lecture de code conditionnelle.

## Description

Si la macro spécifiée n'est pas définie alors *#ifndef* peut incorporer du texte dans un orchestre jusqu'au prochain *#end*. C'est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

## Syntaxe

```
#ifndef NOM  
  
....  
  
#else  
  
....  
  
#end
```

## Exécution

Noter que l'on peut imbriquer les *#ifndef*, comme dans le langage du préprocesseur C.

## Voir Aussi

*\$NOM*, *#define*, *#ifdef*.

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 2005

Nouveau dans Csound5 (et 4.23f13)

# \$NOM

\$NOM — Appelle une macro définie.

## Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et \$ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

*\$NOM* -- appelle une macro définie. Pour appeler une macro, on utilise son nom précédé du caractère \$. La fin du nom est marquée par le premier caractère qui n'est ni une lettre ni un chiffre. S'il est nécessaire que ce caractère ne soit pas un espace, on peut utiliser un point, qui sera ignoré, pour terminer le nom. La chaîne, *\$NOM.*, est remplacée par le texte de substitution de la définition. Le texte de substitution peut lui-même comprendre des appels de macro.

## Syntaxe

\$NOM

## Initialisation

*# texte de substitution #* -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est entouré par des caractères #, ce qui garantit qu'aucun caractère supplémentaire ne sera capturé par inadvertance.

## Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

## Exemples

Voici un exemple d'appel de macro. Il utilise le fichier *define.csd* [examples/define.csd].

### Exemple 13. Un exemple d'appel de macro.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o define.wav -W ;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the macros.
#define VOLUME #5000#
#define FREQ #440#
#define TABLE #1#

; Instrument #1
instr 1
; Use the macros.
; This will be expanded to "a1 oscil 5000, 440, 1".
a1 oscil $VOLUME, $FREQ, $TABLE

; Send it to the output.
out a1
endin

</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie présentera des lignes comme celles-ci :

```
Macro definition for VOLUME
Macro definition for CPS
Macro definition for TABLE
```

Voici un exemple d'appel de macro avec arguments. Il utilise le fichier *define\_args.csd* [exemples/define\_args.csd].

## Exemple 14. Un exemple d'appel de macro avec arguments.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o define_args.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the oscillator macro.
#define OSCMACRO(VOLUME'FREQ'TABLE) #oscil $VOLUME, $FREQ, $TABLE#

; Instrument #1
instr 1
; Use the oscillator macro.
; This will be expanded to "a1 oscil 5000, 440, 1".
```

```
al $OSCMACRO(5000'440'1)

; Send it to the output.
out al
endin

</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

La sortie présentera des lignes comme celle-ci :

```
Macro definition for OSCMACRO
```

## Voir Aussi

*#define, #undef*

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.48 de Csound



# %

% — Opérateur modulo.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

$a + b * c$ .

Trois règles s'appliquent dans de tels cas :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus s'interprète comme

$a + (b * c)$

avec  $*$  prenant  $b$  et  $c$  puis  $+$  prenant  $a$  et  $b * c$ .

2.  $+$  et  $-$  sont prioritaires sur  $\&\&$ , qui devance lui-même  $\|$  :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses pour forcer un groupement particulier.

L'opérateur  $\%$  retourne la valeur de la réduction de  $a$  par  $b$ , de telle façon que le résultat, en valeur absolue, est inférieur à la valeur absolue de  $b$ , par soustraction répétée. C'est l'équivalent de la fonction modulo pour les entiers. Nouveau dans la version 3.50 de Csound.

## Syntaxe

`a % b` (pas de restriction de taux)

où les arguments  $a$  et  $b$  peuvent être des expressions.

## Exemples

Voici un exemple de l'opérateur %. Il utilise le fichier *modulus.csd* [examples/modulus.csd].

### Exemple 15. Exemple de l'opérateur %.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o modulus.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  il = 5 % 3
  print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie présentera une ligne comme celle-ci :

```
instr 1:  il = 2.000
```

## Voir Aussi

-, +, &&, //, \*, /, ^

## Crédits

Exemple écrit par Kevin Conder.

## &&

&& — Opérateur ET logique.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

$a + b * c$ .

Trois règles s'appliquent dans de tels cas :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus s'interprète comme

$a + (b * c)$

avec  $*$  prenant  $b$  et  $c$  puis  $+$  prenant  $a$  et  $b * c$ .

2.  $+$  et  $-$  sont prioritaires sur  $&&$ , qui devance lui-même  $||$  :

$a \&\& b - c || d$

est interprété comme

$(a \&\& (b - c)) || d$

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses pour forcer un groupement particulier.

## Syntaxe

$a \&\& b$  (ET logique ; pas de taux audio)

où les arguments  $a$  et  $b$  peuvent être des expressions.

## Voir Aussi

$-$ ,  $+$ ,  $||$ ,  $*$ ,  $/$ ,  $^$ ,  $\%$

## >

> — Détermine si une valeur est supérieure à l'autre.

## Description

Détermine si une valeur est supérieure à l'autre.

## Syntaxe

```
(a > b ? v1 : v2)
```

où  $a$ ,  $b$ ,  $v1$  et  $v2$  peuvent être des expressions, mais  $a$ ,  $b$  pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus,  $a$  et  $b$  sont d'abord comparés. Si la relation indiquée est vraie ( $a$  supérieur à  $b$ ), alors l'expression conditionnelle prend la valeur de  $v1$  ; si la relation est fausse, l'expression prend la valeur de  $v2$ .

Nota Bene : Si  $v1$  ou  $v2$  sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur >. Il utilise le fichier *greaterthan.csd* [examples/greaterthan.csd].

### Exemple 16. Exemple de l'opérateur >.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o greaterthan.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```

instr 1
; Get the 4th p-field from the score.
k1 = p4

; Is it greater than 3? (1 = true, 0 = false)
k2 = (p4 > 3 ? 1 : 0)

; Print the values of k1 and k2.
printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin

</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

k1 = 2.000000, k2 = 0.000000
k1 = 3.000000, k2 = 0.000000
k1 = 4.000000, k2 = 1.000000

```

## Voir Aussi

`==`, `>=`, `<=`, `<`, `!=`

## Crédits

Exemple écrit par Kevin Conder.

## >=

>= — Détermine si une valeur est supérieure ou égale à l'autre.

## Description

Détermine si une valeur est supérieure ou égale à l'autre.

## Syntaxe

```
(a >= b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* supérieur ou égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En terme de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur >=. Il utilise le fichier *greaterequal.csd* [examples/greaterequal.csd].

### Exemple 17. Exemple de l'opérateur >=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o greaterequal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```

instr 1
; Get the 4th p-field from the score.
k1 = p4

; Is it greater than or equal to 3? (1 = true, 0 = false)
k2 = (p4 >= 3 ? 1 : 0)

; Print the values of k1 and k2.
printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin

</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

k1 = 2.000000, k2 = 0.000000
k1 = 3.000000, k2 = 1.000000
k1 = 4.000000, k2 = 1.000000

```

## Voir Aussi

`==`, `>`, `<=`, `<`, `!=`

## Crédits

Exemple écrit par Kevin Conder.

## &lt;

< — Détermine si une valeur est inférieure à l'autre.

## Description

Détermine si une valeur est inférieure à l'autre.

## Syntaxe

```
( a < b ? v1 : v2 )
```

où  $a$ ,  $b$ ,  $v1$  et  $v2$  peuvent être des expressions, mais  $a$ ,  $b$  pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus,  $a$  et  $b$  sont d'abord comparés. Si la relation indiquée est vraie ( $a$  inférieur à  $b$ ), alors l'expression conditionnelle prend la valeur de  $v1$  ; si la relation est fausse, l'expression prend la valeur de  $v2$ .

Nota Bene : Si  $v1$  ou  $v2$  sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En terme de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur <. Il utilise le fichier *lessthan.csd* [examples/lessthan.csd].

### Exemple 18. Exemple de l'opérateur <.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lessthan.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```



```

instr 1
; Get the 4th p-field from the score.
k1 = p4

; Is it less than 3? (1 = true, 0 = false)
k2 = (p4 < 3 ? 1 : 0)

; Print the values of k1 and k2.
printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin

</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

k1 = 2.000000, k2 = 1.000000
k1 = 3.000000, k2 = 0.000000
k1 = 4.000000, k2 = 0.000000

```

## Voir Aussi

`==`, `>=`, `>`, `<=`, `!=`

## Crédits

Exemple écrit par Kevin Conder.

## <=

<= — Détermine si une valeur est inférieure ou égale à l'autre.

## Description

Détermine si une valeur est inférieure ou égale à l'autre.

## Syntaxe

```
( a <= b ? v1 : v2 )
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* inférieur ou égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur <=. Il utilise le fichier *lessequal.csd* [exemples/lessequal.csd].

### Exemple 19. Exemple de l'opérateur <=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lessequal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```
instr 1
; Get the 4th p-field from the score.
k1 = p4

; Is it less than or equal to 3? (1 = true, 0 = false)
k2 = (p4 <= 3 ? 1 : 0)

; Print the values of k1 and k2.
printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin

</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
k1 = 2.000000, k2 = 1.000000
k1 = 3.000000, k2 = 1.000000
k1 = 4.000000, k2 = 0.000000
```

## Voir Aussi

`==`, `>=`, `>`, `<`, `!=`

## Crédits

Exemple écrit par Kevin Conder.

## \*

\* — Opérateur de multiplication

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1. \* et / s'appliquent à leurs voisins plus fortement que + et -. Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec \* s'appliquant à b et à c et ensuite + s'appliquant à a et à  $b * c$ .

2. + et - sont prioritaires par rapport à &&, qui est lui-même prioritaire par rapport à ||:

$a \&\& b - c \parallel d$

est interprété comme

$(a \&\& (b - c)) \parallel d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

## Syntaxe

$a * b$  (pas de restriction de taux)

où les arguments *a* et *b* peuvent être des expressions.

## Exemples

Voici un exemple de l'opérateur \*. Il utilise le fichier *multiplies.csd* [examples/multiplies.csd].

## Exemple 20. Exemple de l'opérateur \*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o multiplies.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 24 * 8
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 192.000
```

## Voir Aussi

-, +, &&, //, /, ^, %

## Crédits

Exemple écrit par Kevin Conder.

# +

+ — Opérateur d'addition

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $\#$ . Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec  $*$  s'appliquant à  $b$  et à  $c$  et ensuite  $+$  s'appliquant à  $a$  et à  $b * c$ .

2.  $+$  et  $\#$  sont prioritaires par rapport à  $\&\&$ , qui est lui-même prioritaire par rapport à  $\|$ :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

## Syntaxe

`+a` (pas de restriction de taux)

`a + b` (pas de restriction de taux)

où les arguments  $a$  et  $b$  peuvent être des expressions.

## Exemples

Voici un exemple de l'opérateur  $+$ . Il utilise le fichier *adds.csd* [examples/adds.csd].

## Exemple 21. Exemple de l'opérateur +.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o adds.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 24 + 8
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 32.000
```

## Voir Aussi

-, &&, //, \*, /, ^, %

## Crédits

Exemple écrit par Kevin Conder.

■

- — Opérateur de soustraction.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $\#$ . Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec  $*$  s'appliquant à  $b$  et à  $c$  et ensuite  $+$  s'appliquant à  $a$  et à  $b * c$ .

2.  $+$  et  $\#$  sont prioritaires par rapport à  $\&\&$ , qui est lui-même prioritaire par rapport à  $\|$ :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

## Syntaxe

$\#a$  (pas de restriction de taux)

$a \# b$  (pas de restriction de taux)

où les arguments  $a$  and  $b$  peuvent être des expressions.

## Exemples

Voici un exemple de l'opérateur  $\#$ . Il utilise le fichier *subtracts.csd* [examples/subtracts.csd].



## Exemple 22. Exemple de l'opérateur #.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o subtracts.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  il = 24 - 8
  print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  il = 16.000
```

## Voir Aussi

+, &&, //, \*, /, ^, %

## Crédits

Exemple écrit par Kevin Conder.

# /

/ — Opérateur de division.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec  $*$  s'appliquant à  $b$  et à  $c$  et ensuite  $+$  s'appliquant à  $a$  et à  $b * c$ .

2.  $+$  et  $-$  sont prioritaires par rapport à  $\&\&$ , qui est lui-même prioritaire par rapport à  $\|\|$ :

$a \&\& b - c \|\| d$

est interprété comme

$(a \&\& (b - c)) \|\| d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

## Syntaxe

$a / b$  (pas de restriction de taux)

où les arguments  $a$  et  $b$  peuvent être des expressions.

## Exemples

Voici un exemple de l'opérateur  $/$ . Il utilise le fichier *divides.csd* [examples/divides.csd].

### Exemple 23. Exemple de l'opérateur /.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o divides.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  il = 24 / 8
  print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  il = 3.000
```

## Voir Aussi

-, +, &&, //, \*, ^, %

## Crédits

Exemple écrit par Kevin Conder.

**=**

= — Réalise une simple affectation.

## Syntaxe

```
ares = xarg

ires = iarg

kres = karg

ires, ... = iarg, ...

kres, ... = karg, ...
```

## Description

Réalise une simple affectation.

## Initialisation

= (simple affectation) - Met la valeur de l'expression *iarg* (*karg*, *xarg*) dans le résultat nommé. On peut ainsi garder en mémoire le résultat d'une évaluation pour une utilisation ultérieure.

A partir de la version 5.13 les versions de taux-i et de taux-k de l'affectation peuvent prendre un certain nombre de sorties, et un nombre égal ou inférieur d'entrées. S'il y a moins d'entrées, la dernière valeur est répétée le nombre de fois nécessaires.

## Exemples

Voici un exemple de l'opérateur d'affectation. Il utilise le fichier *assign.csd* [examples/assign.csd].

### Exemple 24. Exemple de l'opérateur d'affectation.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o assign.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
; Assign a value to the variable i1.
i1 = 1234

; Print the value of the i1 variable.
print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 1234.000
```

## Voir Aussi

*divz, init, passign, tival*

## Crédits

Exemple écrit par Kevin Conder.

Extension aux valeurs multiples par

Auteur : John ffitch  
Université de Bath, et Codemist Ltd.  
Bath, UK  
Février 2010

Nouveau dans la version 5.13

## ==

== — Teste l'égalité de deux valeurs.

## Description

Teste l'égalité de deux valeurs.

## Syntaxe

```
(a == b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*. (Par commodité, un seul "=" fonctionnera comme "==".)

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur ==. Il utilise le fichier *equals.csd* [examples/equals.csd].

### Exemple 25. Exemple de l'opérateur ==.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o equal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
```

```

instr 1
; Get the 4th p-field from the score.
k1 = p4

; Is it equal to 3? (1 = true, 0 = false)
k2 = (p4 == 3 ? 1 : 0)

; Print the values of k1 and k2.
printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin

</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

k1 = 2.000000, k2 = 0.000000
k1 = 3.000000, k2 = 1.000000
k1 = 4.000000, k2 = 0.000000

```

## Voir Aussi

>=, >, <=, <, !=

## Crédits

Exemple écrit par Kevin Conder.

## ^

^ — Opérateur d'élévation à une puissance.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec  $*$  s'appliquant à  $b$  et à  $c$  et ensuite  $+$  s'appliquant à  $a$  et à  $b * c$ .

2.  $+$  et  $-$  sont prioritaires par rapport à  $\&\&$ , qui est lui-même prioritaire par rapport à  $\|$ :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

L'opérateur  $^$  élève  $a$  à la puissance  $b$ .  $b$  ne doit pas être de taux audio. A utiliser avec précaution car les règles de précedence peuvent ne pas fonctionner correctement. Voir *pow*. (Nouveau dans la version 3.493 de Csound.)

## Syntaxe

`a ^ b (b pas de taux audio)`

où les arguments  $a$  et  $b$  peuvent être des expressions.

## Exemples



Voici un exemple de l'opérateur  $\wedge$ . Il utilise le fichier *raises.csd* [examples/raises.csd].

## Exemple 26. Exemple de l'opérateur $\wedge$ .

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o raises.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 2 ^ 12
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  i1 = 4096.000
```

## Voir Aussi

-, +, &&, //, \*, /, %

## Crédits

Exemple écrit par Kevin Conder.

# ||

|| — Opérateur OU logique.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

$a + b * c$ .

Trois règles s'appliquent dans de tels cas :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus s'interprète comme

$a + (b * c)$

avec  $*$  prenant  $b$  et  $c$  puis  $+$  prenant  $a$  et  $b * c$ .

2.  $+$  et  $-$  sont prioritaires sur  $\&\&$ , qui devance lui-même  $||$  :

$a \&\& b - c || d$

est interprété comme

$(a \&\& (b - c)) || d$

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses pour forcer un groupement particulier.

## Syntaxe

$a || b$  (OU logique ; pas de taux audio)

où les arguments  $a$  et  $b$  peuvent être des expressions.

## Voir Aussi

$-$ ,  $+$ ,  $\&\&$ ,  $*$ ,  $/$ ,  $^$ ,  $\%$

# Odbfs

Odbfs — Fixe la valeur des 0 décibels à amplitude maximale.

## Description

Fixe la valeur des 0 décibels à amplitude maximale.

## Syntaxe

```
odbfs = iarg
```

```
odbfs
```

## Initialisation

*iarg* -- la valeur des 0 décibels à amplitude maximale.

## Exécution

La valeur par défaut est 32767, si bien que tous les orchestres existants *devraient* fonctionner.

Les valeurs d'amplitude dans Csound sont toujours relatives à une valeur "odbfs" représentant l'amplitude maximale possible sans écrêtage. A l'origine cette valeur valait toujours 32767 dans Csound, correspondant à l'étendue bipolaire d'un fichier son sur 16 bit ou d'un codec AN/NA sur 16 bit. Cela reste l'amplitude maximale *par défaut* dans Csound, pour des raisons de compatibilité descendante. La valeur *odbfs* permet à Csound de produire des valeurs mises à l'échelle de n'importe quel format de sortie, que ce soit en entiers sur 16 bit ou 24 bit, en flottants sur 32 bit, et même en entiers sur 32 bit.

On peut définir Odbfs dans l'en-tête, pour fixer l'amplitude de référence utilisée par Csound, mais on peut aussi l'utiliser comme variable dans un instrument comme ceci :

```
ipeak = odbfs
```

```
asig oscil odbfs, freq, 1  
out asig * 0.3 * odbfs
```

L'opcode *odbfs* a pour but le codage relatif à une valeur *odbfs* (et l'usage beaucoup plus fréquent des opcodes *ampdbfs*() !), plutôt que l'utilisation de valeurs d'échantillon explicites. L'utilisation de *odbfs*=1 est conforme aux pratiques de l'industrie, car l'intervalle allant de -1 à 1 est utilisé dans la plupart des formats de plugin commerciaux et dans la plupart des autres systèmes de synthèse comme Pure Data.

Les nombres en virgule flottante écrits dans un fichier, lorsque *odbfs* = 1, ne seront effectivement pas transposés du tout en amplitude. Ainsi les nombres dans le fichier sont exactement ce que l'orchestre dit qu'ils sont.

Pour plus de détails sur les valeurs d'amplitude dans Csound, voir la section *Valeurs d'amplitude dans Csound*.

## Exemples

Voici un exemple de l'opcode `Odbfs`. Il utilise le fichier `Odbfs.csd` [examples/Odbfs.csd].

### Exemple 27. Exemple de l'opcode `Odbfs`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o Odbfs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Set the Odbfs to the 16-bit maximum.
Odbfs = 32767

; Instrument #1.
instr 1
; Linearly increase the amplitude value "kamp" from
; 0 to 1 over the duration defined by p3.
kamp line 0, p3, 1

; Generate a basic tone using our amplitude value.
a1 oscil kamp, 440, 1

; Multiply the basic tone (with its amplitude between
; 0 and 1) by the full-scale Odbfs value.
out a1 * Odbfs
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode `Odbfs`. Il utilise le fichier `Odbfs-1.csd` [examples/Odbfs-1.csd]. Cet exemple a exactement la même sortie que l'exemple précédent, mais les échantillons de sortie sont maintenant normalisés entre -1 et 1.

### Exemple 28. Exemple de l'opcode `Odbfs` avec une amplitude maximale de 1.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o 0dbfs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Set the 0dbfs to 1.
0dbfs = 1

; Instrument #1.
instr 1
; Linearly increase the amplitude value "kamp" from
; -90 to p4 (in dBfs) over the duration defined by p3.
kamp line -90, p3, p4
print ampdbfs(p4)
; Generate a basic tone using our amplitude value.
a1 oscil ampdbfs(kamp), 440, 1

; Since 0dbfs = 1 we don't need to multiply the output
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3 -6
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*ampdbfs()*

## Crédits

Auteur : Richard Dobson  
Mai 2002

Nouveau dans la version 4.10

Exemple écrit par Kevin Conder.

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.20



<< — Bitshift left operator.

## Description

The bitshift operators shift the bits to the left or to the right the number of bits given.

The priority of these operators is less binding than the arithmetic ones, but more binding than the comparisons.

Parentheses may be used as above to force particular groupings.

## Syntax

```
a << b (bitshift left)
```

where the arguments *a* and *b* may be further expressions.

## Examples

Here is an example of the bitshift left operator. It uses the file *bitshift.csd* [examples/bitshift.csd].

### Exemple 29. Example of the bitshift left operator.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
;-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
-o bitshift.wav -W --nosound ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

instr 1 ;bit shift right
ival = p4>>p5
printf_i "%i>>%i = %i\n", 1, p4, p5, ival
endin

instr 2 ;bit shift left
ival = p4<<p5
printf_i "%i<<%i = %i\n", 1, p4, p5, ival
endin

</CsInstruments>
<CsScore>
i 1 0 0.1 2 1
i 1 + . 3 1
i 1 + . 7 2
i 1 + . 16 1
i 1 + . 16 2
i 1 + . 16 3
```

```

i 2 5 0.1 1 1
i 2 + . 1 2
i 2 + . 1 3
i 2 + . 1 4
i 2 + . 2 1
i 2 + . 2 2
i 2 + . 2 3
i 2 + . 3 2
e
</CsScore>
</CsoundSynthesizer>

```

The example above will produce the following output:

```

2>>1 = 1
B 0.000 .. 0.100 T 0.100 TT 0.100 M: 0.0 0.0
3>>1 = 1
B 0.100 .. 0.200 T 0.200 TT 0.200 M: 0.0 0.0
7>>2 = 1
B 0.200 .. 0.300 T 0.300 TT 0.300 M: 0.0 0.0
16>>1 = 8
B 0.300 .. 0.400 T 0.400 TT 0.400 M: 0.0 0.0
16>>2 = 4
B 0.400 .. 0.500 T 0.500 TT 0.500 M: 0.0 0.0
16>>3 = 2
B 0.500 .. 5.000 T 5.000 TT 5.000 M: 0.0 0.0
new alloc for instr 2:
1<<1 = 2
B 5.000 .. 5.100 T 5.100 TT 5.100 M: 0.0 0.0
1<<2 = 4
B 5.100 .. 5.200 T 5.200 TT 5.200 M: 0.0 0.0
1<<3 = 8
B 5.200 .. 5.300 T 5.300 TT 5.300 M: 0.0 0.0
1<<4 = 16
B 5.300 .. 5.400 T 5.400 TT 5.400 M: 0.0 0.0
2<<1 = 4
B 5.400 .. 5.500 T 5.500 TT 5.500 M: 0.0 0.0
2<<2 = 8
B 5.500 .. 5.600 T 5.600 TT 5.600 M: 0.0 0.0
2<<3 = 16
B 5.600 .. 5.700 T 5.700 TT 5.700 M: 0.0 0.0
3<<2 = 12

```

## See Also

>>, &, /#

## >>

>> — Bitshift right operator.

## Description

The bitshift operators shift the bits to the left or to the right the number of bits given.

The priority of these operators is less binding than the arithmetic ones, but more binding than the comparisons.

Parentheses may be used as above to force particular groupings.

## Syntax

```
a >> b (bitshift left)
```

where the arguments *a* and *b* may be further expressions.

## Examples

See the entry for the << operator for an example.

## See Also

<<, &, / #



# &

& — Opérateur ET binaire.

## Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

## Syntaxe

`a & b` (ET binaire)

où les arguments *a* et *b* peuvent être des expressions. Ils sont convertis à la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

## Exécution

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

## Exemples

Voici un exemple des opérateurs binaires ET et OU. Il utilise le fichier *bitwise.csd* [exemples/bitwise.csd].

### Exemple 30. Exemple des opérateurs binaires.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>

</CsOptions>
<CsInstruments>

instr 1
iresultOr = p4 | p5
iresultAnd = p4 & p5
prints "%i | %i = %i\\n", p4, p5, iresultOr
prints "%i & %i = %i\\n", p4, p5, iresultAnd
endin

instr 2 ; decimal to binary converter
Sbinary = ""
inumbits = 8
icount init inumbits - 1

pass:

    ivalue = 2 ^ icount
    if (p4 & ivalue >= ivalue) then
        Sdigit = "1"
```

```

        else
            Sdigit = "0"
        endif
        Sbinary strcat Sbinary, Sdigit
    loop_ge icount, 1, 0, pass

    Stext sprintf "%i is %s in binary\\n", p4, Sbinary
    prints Stext
endin

</CsInstruments>
<CsScore>
i 1 0 0.1 1 2
i 1 + . 1 3
i 1 + . 2 4
i 1 + . 3 10

i 2 2 . 12
i 2 + . 9
i 2 + . 15
i 2 + . 49

e
</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

/, #, ¬

|

| — Opérateur OU binaire.

## Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

## Syntaxe

$a \mid b$  (OU binaire)

où les arguments  $a$  et  $b$  peuvent être des expressions. Ils sont convertis à la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

## Exécution

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

Pour un exemple d'utilisation, voir l'entrée pour  $\&$

## Voir Aussi

$\&$ ,  $\#$ ,  $\neg$

¬

¬ — Opérateur NON binaire.

## Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

## Syntaxe

`~ a (NON binaire)`

où l'argument *a* peut être une expression. Il est converti dans la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

## Voir Aussi

`&, / #`

# #

# — Opérateur NON-EQUIVALENCE binaire.

## Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

## Syntaxe

`a # b` (NON-EQUIVALENCE binaire)

où les arguments  $a$  et  $b$  peuvent être des expressions. Ils sont convertis dans la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

## Voir Aussi

$\&$ ,  $/$ ,  $\neg$

## a

a — Convertit un paramètre de taux-k en une valeur de taux-a avec interpolation.

## Description

Convertit un paramètre de taux-k en une valeur de taux-a avec interpolation.

## Syntaxe

**a**(x) (arguments de taux-k seulement)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode a. Il utilise le fichier *opa.csd* [examples/opa.csd].

### Exemple 31. Exemple de l'opcode a.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o a.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create a sine wave at k-rate.
kwave oscil 20000, 440, 1

; Convert the k-rate sine wave to the audio-rate.
awave = a(kwave)

; Output the audio-rate version of sine wave.
out awave
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
```

**i** 1 0 1  
**e**

</CsScore>  
</CsoundSynthesizer>

## Voir Aussi

*i, k*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.21

# abetarand

abetarand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.



# abexprnd

abexprnd — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.

# abs

abs — Retourne une valeur absolue.

## Description

Retourne la valeur absolue de  $x$ .

## Syntaxe

**abs**( $x$ ) (pas de restriction de taux)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode abs. Il utilise le fichier *abs.csd* [examples/abs.csd].

### Exemple 32. Exemple de l'opcode abs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
i1 = -6
i2 = abs(i1)

print i2
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie incluera des lignes comme :

```
instr 1: i2 = 6.000
```

## Voir Aussi

*exp, frac, int, log, log10, i, sqrt*

## Crédits

Exemple écrit par Kevin Conder.

# acauchy

acauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

# active

active — Retourne le nombre d'instances actives d'un instrument.

## Description

Retourne le nombre d'instances actives d'un instrument.

## Syntaxe

```
ir active insnum
```

```
kres active kinsnum
```

## Initialisation

*insnum* -- numéro de l'instrument concerné

## Exécution

*kinsnum* -- numéro de l'instrument concerné

*active* retourne le nombre d'instances actives de l'instrument numéro *insnum/kinsnum*. A partir de Csound 4.17 la sortie est mise à jour au taux-k (si l'argument d'entrée est de taux-k), pour permettre un comptage dynamique des instances d'instrument.

## Exemples

Voici un exemple de l'opcode active. Il utilise le fichier *active.csd* [examples/active.csd].

### Exemple 33. Exemple simple de l'opcode active.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o active.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a noisy waveform.
instr 1
; Generate a really noisy waveform.
anoisy rand 44100
```

```

; Turn down its amplitude.
aoutput gain anois, 2500
; Send it to the output.
out aoutput
endin

; Instrument #2 - counts active instruments.
instr 2
; Count the active instances of Instrument #1.
icount active 1
; Print the number of active instances.
print icount
endin

</CsInstruments>
<CsScore>

; Start the first instance of Instrument #1 at 0:00 seconds.
i 1 0.0 3.0

; Start the second instance of Instrument #1 at 0:015 seconds.
i 1 1.5 1.5

; Play Instrument #2 at 0:01 seconds, when we have only
; one active instance of Instrument #1.
i 2 1.0 0.1

; Play Instrument #2 at 0:02 seconds, when we have
; two active instances of Instrument #1.
i 2 2.0 0.1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

instr 2: icount = 1.000
instr 2: icount = 2.000

```

Voici un exemple plus avancé de l'opcode active. Il affiche le résultat de l'opcode active au taux-k. Il utilise le fichier *active\_k.csd* [examples/active\_k.csd].

### Exemple 34. Exemple de l'opcode active au taux-k.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in
-odac -iadc ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o active_k.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a noisy waveform.
instr 1
; Generate a really noisy waveform.
anoisy rand 44100
; Turn down its amplitude.
aoutput gain anois, 2500
; Send it to the output.
out aoutput
endin

```

```

; Instrument #2 - counts active instruments at k-rate.
instr 2
; Count the active instances of Instrument #1.
kcount active 1
; Print the number of active instances.
printk2 kcount
endin

</CsInstruments>
<CsScore>

; Start the first instance of Instrument #1 at 0:00 seconds.
i 1 0.0 3.0

; Start the second instance of Instrument #1 at 0:015 seconds.
i 1 1.5 1.5

; Play Instrument #2 at 0:01 seconds, when we have only
; one active instance of Instrument #1.
i 2 1.0 0.1

; Play Instrument #2 at 0:02 seconds, when we have
; two active instances of Instrument #1.
i 2 2.0 0.1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

i2      1.00000
i2      2.00000

```

Voici un autre exemple de l'opcode active, qui utilise le nombre d'instances pour calculer le gain. Il utilise le fichier *active\_scale.csd* [examples/active\_scale.csd].

### Exemple 35. Exemple de l'opcode active au taux-k.

```

<CsoundSynthesizer>
<CsOptions>

</CsOptions>
<CsInstruments>

sr= 44100
ksmps = 64
0dbfs = 1

;by Victor Lazzarini 2008

instr 1
kscal active 1
kamp port 1/kscal, 0.01
asig oscili kamp, p4, 1
kenv linseg 0, 0.1,1,p3-0.2,1,0.1, 0

      out asig*kenv
endin

</CsInstruments>
<CsScore>
f1 0 16384 10 1

i1 0 10 440
i1 1 3 220
i1 2 5 350
i1 4 3 700
</CsScore>

```

`</CsoundSynthesizer>`

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Juillet 1999

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.57 de Csound



## adsr

`adsr` — Calcule l'enveloppe ADSR classique à l'aide de segments linéaires.

## Description

Calcule l'enveloppe ADSR classique à l'aide de segments linéaires.

## Syntaxe

```
ares adsr iatt, idec, islev, irel [, idel]
```

```
kres adsr iatt, idec, islev, irel [, idel]
```

## Initialisation

*iatt* -- durée de l'attaque (attack)

*idec* -- durée de la première chute (decay)

*islev* -- niveau d'entretien (sustain)

*irel* -- durée de la chute (release)

*idel* -- délai de niveau zéro avant le démarrage de l'enveloppe

## Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

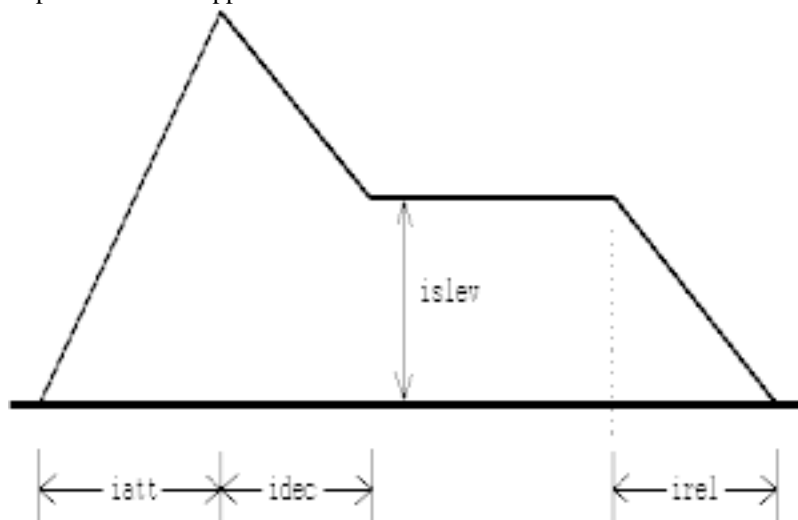


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des événements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et

peut donc être utilisé dans les applications MIDI.

*adsr* est nouveau dans la version 3.49 de Csound.

## Exemples

Voici un exemple de l'opcode *adsr*. Il utilise le fichier *adsr.csd* [examples/adsr.csd].

### Exemple 36. Exemple de l'opcode *adsr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o adsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a simple instrument.
instr 1
; Set the amplitude.
kamp init 20000
; Get the frequency from the fourth p-field.
kcps = cpspch(p4)

al vco kamp, kcps, 1
out al
endin

; Instrument #2 - instrument with an ADSR envelope.
instr 2
iatt = 0.05
idec = 0.5
islev = 0.08
irel = 0.008

; Create an amplitude envelope.
kenv adsr iatt, idec, islev, irel
kamp = kenv * 20000

; Get the frequency from the fourth p-field.
kcps = cpspch(p4)

al vco kamp, kcps, 1
out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Set the tempo to 120 beats per minute.
t 0 120

; Play a melody with Instrument #1.
; p4 = frequency in pitch-class notation.
i 1 0 1 8.04
i 1 1 1 8.04
i 1 2 1 8.05
```

```

i 1 3 1 8.07
i 1 4 1 8.07
i 1 5 1 8.05
i 1 6 1 8.04
i 1 7 1 8.02
i 1 8 1 8.00
i 1 9 1 8.00
i 1 10 1 8.02
i 1 11 1 8.04
i 1 12 2 8.04
i 1 14 2 8.02

; Repeat the melody with Instrument #2.
; p4 = frequency in pitch-class notation.
i 2 16 1 8.04
i 2 17 1 8.04
i 2 18 1 8.05
i 2 19 1 8.07
i 2 20 1 8.07
i 2 21 1 8.05
i 2 22 1 8.04
i 2 23 1 8.02
i 2 24 1 8.00
i 2 25 1 8.00
i 2 26 1 8.02
i 2 27 1 8.04
i 2 28 2 8.04
i 2 30 2 8.02
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*madsr, mxadsr, xadsr*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.49

Exemple écrit par Kevin Conder.

# adsyn

adsyn — La sortie est la somme d'un ensemble de sinusôides contrôlées individuellement, jouées par un banc d'oscillateurs.

## Description

La sortie est la somme d'un ensemble de sinusôides contrôlées individuellement, jouées par un banc d'oscillateurs.

## Syntaxe

ares **adsyn** kamod, kfmod, ksmod, ifilcod

## Initialisation

*ifilcod* -- entier ou chaîne de caractères dénotant un fichier de contrôle issu de l'analyse d'un signal audio. Un entier dénote le suffixe d'un fichier *adsyn.m* ou *pvoc.m* ; une chaîne de caractères (entre doubles apostrophes) donne un nom de fichier, optionnellement un nom de chemin complet. S'il ne s'agit pas d'un chemin complet, le fichier est d'abord recherché dans le répertoire courant, puis dans celui qui est indiqué par la variable d'environnement *SADIR* (si elle est définie). Le fichier de contrôle *adsyn* contient les valeurs des points charnière des enveloppes d'amplitude et de fréquence, tandis que le fichier de contrôle *pvoc* contient des données similaires organisées pour une resynthèse par tfr. L'utilisation de la mémoire dépend de la taille des fichiers impliqués, qui sont lus et maintenus entièrement en mémoire durant le calcul tout en étant partagés par les appels multiples (voir aussi *lpread*).

## Exécution

*kamod* -- facteur d'amplitude des partiels additionnés.

*kfmod* -- facteur de fréquence des partiels additionnés. C'est un facteur de transposition au taux de contrôle : une valeur de 1 signifie pas de transposition, 1,5 transpose d'un quinte juste ascendante, et 0,5 d'une octave descendante.

*ksmod* -- facteur de vitesse des partiels additionnés.

*adsyn* synthétise des timbres dynamiques complexes par la méthode de synthèse additive. N'importe quel nombre de sinusôides, contrôlées individuellement en fréquence et en amplitude, peuvent être additionnées par une unité arithmétique très rapide pour produire un résultat de grande qualité.

Les composantes sinusoidales sont décrites dans un fichier de contrôle qui contient des pistes d'amplitude et de fréquence définies par des points charnière. Les pistes sont des séquences de nombres entiers sur 16 bit :

-1, date, amp, date, amp,...

-2, date, fréq, date, fréq,...

telles que celles qui sont produites par l'analyse d'un fichier audio au moyen d'un filtre hétérodyne. (Pour des détails, voir *hetro*.) Les valeurs instantanées d'amplitude et de fréquence sont utilisées par un oscillateur interne en virgule fixe qui additionne chaque partiel actif dans un signal de sortie accumulé. Bien qu'il y ait une limite pratique (limite supprimée dans la version 3.47) du nombre de partiels mis à contribution, il n'y a aucune restriction quant à leur comportement dans le temps. Un son quelconque que l'on

peut décrire en termes d'évolution de sinusoïdes sera synthétisable par *adsyn* seul.

On peut aussi modifier un son décrit par un fichier de contrôle *adsyn* pendant la resynthèse. Les signaux *kamod*, *kfmod* et *ksmod* modifieront l'amplitude, la fréquence et la vitesse des partiels. Ce sont des facteurs multiplicatifs, avec *kfmod* modifiant la fréquence et *ksmod* modifiant la *vitesse* avec laquelle les segments en millisecondes définis par les points charnière sont parcourus. Ainsi, 0,7, 1,5 et 2 produiront un son plus doux, plus haut d'une quinte juste, mais deux fois moins long. Les valeurs 1, 1, 1 laisseront le son inchangé. Chacune de ces entrées peut être un signal de contrôle.

## Exemples

Voici un exemple de l'opcode *adsyn*. Il utilise les fichiers *adsyn.csd* [examples/adsyn.csd] et *kickroll.het* [examples/kickroll.het]. Le fichier « *kickroll.het* » a été créé en utilisant l'utilitaire *hetro* avec le fichier audio *kickroll.wav* [examples/kickroll.wav].

### Exemple 37. Exemple de l'opcode *adsyn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac          -iadc          -d          ;;RT audio I/O
; -o adsyn.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1
0dbfs = 1

; Instrument #1.
instr 1
; If the modulation amounts are set to 1, adsyn
; will not perform any special modulation.
kamod init 1
kfmod init 1
ksmod init 1

; Re-synthesizes the file "kickroll.het".
a1 adsyn kamod, kfmod, ksmod, "kickroll.het"

out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Exemple écrit par Kevin Conder.

# adsynt

adsynt — Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques.

## Description

Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques.

## Syntaxe

```
ares adsynt kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

## Initialisation

*iwfn* -- table contenant une forme d'onde, normalement une sinus. Les valeurs de la table ne sont pas interpolées pour des raisons de performance, si bien que des tables plus grandes apportent une meilleure qualité.

*ifreqfn* -- table contenant les valeurs de fréquence de chaque partiel. *ifreqfn* peut contenir les valeurs de fréquence initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les fréquences doivent être relatives à *kcps*. La taille doit être au moins égale à *icnt*.

*iampfn* -- table contenant les valeurs d'amplitude de chaque partiel. *iampfn* peut contenir les valeurs d'amplitude initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les amplitudes doivent être relatives à *kamp*. La taille doit être au moins égale à *icnt*.

*icnt* -- nombre de partiels à générer.

*iphs* -- phase initiale de chaque oscillateur, si *iphs* = -1, l'initialisation est ignorée. Si *iphs* > 1, toutes les phases seront initialisées avec une valeur aléatoire.

## Exécution

*kamp* -- amplitude de la note.

*kcps* -- fréquence de base de la note. Les fréquences des partiels seront relatives à *kcps*.

La fréquence et l'amplitude de chaque partiel sont données dans les deux tables fournies. Le but de cet opcode est de faire générer par un instrument les paramètres de synthèse au taux-k et de les écrire dans des tables globales avec l'opcode *tablew*.

## Exemples

Voici un exemple de l'opcode *adsynt*. Il utilise le fichier *adsynt.csd* [examples/adsynt.csd]. Ces deux instruments réalisent une synthèse additive. La sortie de chacun d'entre eux sonne comme un bol tibétain. Le premier est statique, car ses paramètres ne sont générés que pendant l'initialisation. Dans le second, les paramètres changent de façon continue.

## Exemple 38. Exemple de l'opcode adsynt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o adsynt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Generate a sinewave table.
giwave ftgen 1, 0, 1024, 10, 1
; Generate two empty tables for adsynt.
gifrqs ftgen 2, 0, 32, 7, 0, 32, 0
; A table for frequency and amp parameters.
giamps ftgen 3, 0, 32, 7, 0, 32, 0

; Generates parameters at init time
instr 1
; Generate 10 voices.
icnt = 10
; Init loop index.
index = 0

; Loop only executed at init time.
loop:
; Define non-harmonic partials.
ifreq pow index + 1, 1.5
; Define amplitudes.
iamp = 1 / (index+1)
; Write to tables.
tableiw ifreq, index, gifrqs
; Used by adsynt.
tableiw iamp, index, giamps

index = index + 1
; Do loop/
if (index < icnt) igoto loop

asig adsynt 5000, 150, giwave, gifrqs, giamps, icnt
out asig
endin

; Generates parameters every k-cycle.
instr 2
; Generate 10 voices.
icnt = 10
; Reset loop index.
kindex = 0

; Loop executed every k-cycle.
loop:
; Generate lfo for frequencies.
kspeed pow kindex + 1, 1.6
; Individual phase for each voice.
kphas phasorbnk kspeed * 0.7, kindex, icnt
klfo table kphas, giwave, 1
; Arbitrary parameter twiddling...
kdepth pow 1.4, kindex
kfreq pow kindex + 1, 1.5
kfreq = kfreq + klfo*0.006*kdepth

; Write freqs to table for adsynt.
tablew kfreq, kindex, gifrqs
```

```

; Generate lfo for amplitudes.
kspeed pow kindex + 1, 0.8
; Individual phase for each voice.
kphas phasorbnk kspeed*0.13, kindex, icnt, 2
klfo table kphas, giwave, 1
; Arbitrary parameter twiddling...
kamp pow 1 / (kindex + 1), 0.4
kamp = kamp * (0.3+0.35*(klfo+1))

; Write amps to table for adsynt.
tablew kamp, kindex, giamps

kindex = kindex + 1
; Do loop.
if (kindex < icnt) kgoto loop

asig adsynt 5000, 150, giwave, gifrqs, giamps, icnt
out asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2.5 seconds.
i 1 0 2.5
; Play Instrument #2 for 2.5 seconds.
i 2 3 2.5
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Peter Neubäcker  
Munich, Allemagne  
Août 1999

Nouveau dans la version 3.58 de Csound



## adsynt2

adsynt2 — Réalise une synthèse additive avec un nombre arbitraire de partiels - pas nécessairement harmoniques - avec interpolation.

### Description

Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques. (Voir *adsynt*)

### Syntaxe

```
ar adsynt2 kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

### Initialisation

*iwfn* -- table contenant une forme d'onde, normalement une sinus. Les valeurs de la table ne sont pas interpolées pour des raisons de performance, si bien que des tables plus grandes apportent une meilleure qualité.

*ifreqfn* -- table contenant les valeurs de fréquence de chaque partiel. *ifreqfn* peut contenir les valeurs de fréquence initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les fréquences doivent être relatives à *kcps*. La taille doit être au moins égale à *icnt*.

*iampfn* -- table contenant les valeurs d'amplitude de chaque partiel. *iampfn* peut contenir les valeurs d'amplitude initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les amplitudes doivent être relatives à *kamp*. La taille doit être au moins égale à *icnt*.

*icnt* -- nombre de partiels à générer.

*iphs* -- phase initiale de chaque oscillateur, si *iphs* = -1, l'initialisation est ignorée. Si *iphs* > 1, toutes les phases seront initialisées avec une valeur aléatoire.

### Exécution

*kamp* -- amplitude de la note.

*kcps* -- fréquence de base de la note. Les fréquences des partiels seront relatives à *kcps*.

La fréquence et l'amplitude de chaque partiel sont données dans les deux tables fournies. Le but de cet opcode est de faire générer par un instrument les paramètres de synthèse au taux-k et de les écrire dans des tables globales avec l'opcode *tablew*.

*adsynt2* est identique à *adsynt* (by Peter Neubäcker), sauf qu'il réalise une interpolation linéaire pour les enveloppes d'amplitude de chaque partiel. Il est un peu plus lent que *adsynt*, mais l'interpolation améliore grandement la qualité du son dans les transitoires rapides des enveloppes d'amplitude lorsque  $kr < sr$  (c'est-à-dire quand  $ksmps > 1$ ). Il n'y a pas d'interpolation pour les enveloppes de hauteur, car dans ce cas la dégradation de la qualité sonore n'est pas aussi évidente même avec de grandes valeurs de *ksmps*. Il n'est pas recommandé quand  $kr = sr$  ; dans ce cas, *adsynt* est meilleur (car plus rapide).

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Disponible auparavant seulement dans CsoundAV)

# **aexprand**

aexprand — Obsolète.

## **Description**

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.

# aftouch

aftouch — Reçoit la valeur d'after-touch actuelle de ce canal.

## Description

Reçoit la valeur d'after-touch actuelle de ce canal.

## Syntaxe

```
kaft aftouch [imin] [, imax]
```

## Initialisation

*imin* (facultatif, par défaut 0) -- limite minimale des valeurs obtenues.

*imax* (facultatif, par défaut 127) -- limite maximale des valeurs obtenues.

## Exécution

Reçoit la valeur d'after-touch actuelle de ce canal.

## Exemples

Voici un exemple de l'opcode aftouch. Il utilise le fichier *aftouch.csd* [examples/aftouch.csd].

### Exemple 39. Exemple de l'opcode aftouch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc       -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o aftouch.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  k1 aftouch

  printk2 k1
endin

</CsInstruments>
<CsScore>
```

```
; Play Instrument #1 for 12 seconds.  
i 1 0 12  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteurs : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par Kevin Conder.

# agauss

agauss — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.

# agogobel

agogobel — Obsolète.

## Description

Nouveau dans la version 3.47

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *gogobel*.

# alinrand

alinrand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.



# alpass

alpass — Réverbère un signal en entrée avec une réponse en fréquence plate.

## Description

Réverbère un signal en entrée avec une réponse en fréquence plate.

## Syntaxe

```
ares alpass asig, krvt, ilpt [, iskip] [, insmps]
```

## Initialisation

*ilpt* -- durée de boucle en secondes, déterminant la « densité d'écho » de la réverbération. Cela caractérise aussi la « couleur » du filtre dont la courbe de réponse en fréquence contiendra  $ilpt * sr/2$  sommets régulièrement espacés entre 0 et  $sr/2$  (la fréquence de Nyquist). La durée de boucle n'est limitée que par la quantité de mémoire disponible. L'espace requis pour une boucle de  $n$  secondes est de  $4n*sr$  octets. L'espace de retard est alloué et retourné comme dans *delay*.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données de la boucle de retard (cf. *reson*). La valeur par défaut est 0.

*insmps* (facultatif, par défaut 0) -- importance du retard, en nombre d'échantillons.

## Exécution

*krvt* -- la durée de réverbération (définie comme le temps en secondes pris par un signal pour faire décroître son amplitude à 1/1000, ou 60 dB de son amplitude originale).

Ce filtre réitère l'entrée avec une densité d'écho déterminée par la durée de boucle *ilpt*. La vitesse d'atténuation est indépendante et déterminée par *krvt*, la durée de réverbération (définie comme le temps en secondes pris par un signal pour faire décroître son amplitude à 1/1000, ou 60 dB de son amplitude originale). La sortie apparaît sans retard.

## Exemples

Voici un exemple de l'opcode alpass. Il utilise le fichier *alpass.csd* [examples/alpass.csd].

### Exemple 40. Exemple de l'opcode alpass.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o alpass.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the audio mixer.
gamix init 0

; Instrument #1.
instr 1
; Generate a source signal.
a1 oscili 30000, cpspch(p4), 1
; Output the direct sound.
out a1

; Add the source signal to the audio mixer.
gamix = gamix + a1
endin

; Instrument #99 (highest instr number executed last)
instr 99
krvt = 1.5
ilpt = 0.1

; Filter the mixed signal.
a99 alpass gamix, krvt, ilpt
; Output the result.
out a99

; Empty the mixer for the next pass.
gamix = 0
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; p4 = frequency (in a pitch-class)
; Play Instrument #1 for a tenth of a second, p4=7.00
i 1 0 0.1 7.00
; Play Instrument #1 for a tenth of a second, p4=7.02
i 1 1 0.1 7.02
; Play Instrument #1 for a tenth of a second, p4=7.04
i 1 2 0.1 7.04
; Play Instrument #1 for a tenth of a second, p4=7.06
i 1 3 0.1 7.06

; Make sure the filter remains active.
i 99 0 5
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*comb, reverb, valpass, vcomb*

## Crédits

Auteur : William « Pete » Moss (*vcomb* et *valpass*)  
 Université du Texas à Austin  
 Austin, Texas USA  
 Janvier 2002

Exemple écrit par Kevin Conder.

# alwayson

**alwayson** — Activates the indicated instrument in the orchestra header, without need for an `i` statement. Instruments must be activated in the same order as they are defined.

## Description

Activates the indicated instrument in the orchestra header, without need for an `i` statement. Instruments must be activated in the same order as they are defined.

The `alwayson` opcode is useful in conjunction with the signal flow graph opcodes. The opcode enables the use of instruments as effects processors that do not have to be turned on in the score, that can receive the outputs of a set or subset of instruments, and that can be chained together.

## Syntax

```
alwayson Tinstrument [p4, ..., pn]
```

## Initialization

*Tinstrument* -- String name, or number, of the instrument to be activated.

*Optional arguments* -- Instrument pfields from `p4` on may be passed to the instrument.

## Performance

The activated instrument will remain "on" for the duration of the performance, receiving signals in any of its inlets and sending signals out through any of its outlets.



### Order is important

The order of definition of instruments is important. A source instrument must be defined in the orchestra before any of the sink instruments to which its outlets are connected.

## See Also

*outleta outletk outletf inleta inletk inletf alwayson ftgenonce*

## Credits

By: Michael Gogins 2009

# ampdb

ampdb — Retourne l'amplitude équivalente à la valeur  $x$  donnée en décibel.

## Description

Retourne l'amplitude équivalente à la valeur  $x$  donnée en décibel.

- 60 dB = 1000
- 66 dB = 1995.262
- 72 dB = 3891.07
- 78 dB = 7943.279
- 84 dB = 15848.926
- 90 dB = 31622.764

## Syntaxe

`ampdb(x)` (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode ampdb. Il utilise le fichier *ampdb.csd* [exemples/ampdb.csd].

### Exemple 41. Exemple de l'opcode ampdb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ampdb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  idb = 90
  iamp = ampdb(idb)

  print iamp
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  iamp = 31622.764
```

## Voir Aussi

*ampdbfs, db, dbamp, dbfsamp*

## Crédits

Exemple écrit par Kevin Conder.

# ampdbfs

ampdbfs — Retourne l'amplitude équivalente (sur une échelle d'entiers signés sur 16 bit) à la valeur  $x$  de l'amplitude maximale (dB FS).

## Description

Retourne l'amplitude équivalente à la valeur  $x$  de l'amplitude maximale (dB FS). Les valeurs logarithmiques de l'échelle en décibels sont converties en valeurs linéaires entières sur 16 bit allant de -32768 à +32767.

## Syntaxe

**ampdbfs**( $x$ ) (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode ampdbfs. Il utilise le fichier *ampdbfs.csd* [examples/ampdbfs.csd].

### Exemple 42. Exemple de l'opcode ampdbfs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ampdbfs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  idb = -1
  iamp = ampdbfs(idb)

  print iamp
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  iamp = 29203.621
```

## Voir Aussi

*ampdb, dbamp, dbfsamp, 0dbfs*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.10 de Csound.



# ampmidi

ampmidi — Retourne la vélocité de l'évènement MIDI en cours.

## Description

Retourne la vélocité de l'évènement MIDI en cours.

## Syntaxe

```
iamp ampmidi iscal [, ifn]
```

## Initialisation

*iscal* -- facteur de pondération de taux-i

*ifn* (facultatif, par défaut 0) -- numéro d'une table de fonction contenant un tableau de conversion normalisé, grâce auquel la valeur entrante est interprétée. La valeur par défaut est 0, ce qui signifie pas de conversion.

## Exécution

Réçoit la vélocité de l'évènement MIDI en cours, la modifie éventuellement grâce à une table de conversion normalisée, et retourne une valeur d'amplitude dans l'intervalle 0 - *iscal*.

## Exemples

Voici un exemple de l'opcode *ampmidi*. Il utilise le fichier *ampmidi.csd* [examples/ampmidi.csd].

### Exemple 43. Exemple de l'opcode *ampmidi*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac        -iadc      -d          -M0    ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o ampmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Scale the amplitude between 0 and 1.
; This example expects MIDI note inputs on channel 1
il ampmidi 1
```

```
    print i1
  endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*aftouch, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteurs : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par Kevin Conder.

# apcauchy

apcauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

# apoisson

apoisson — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

# apow

apow — Obsolète.

## Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

## areson

*areson* — Un filtre réjecteur de bande réglable (notch filter) dont les fonctions de transfert sont les complémentaires de celles de l'opcode *reson*.

## Description

Un filtre réjecteur de bande réglable dont les fonctions de transfert sont les complémentaires de celles de l'opcode *reson*.

## Syntaxe

```
ares areson asig, kcf, kbw [, iscl] [, iskip]
```

## Initialisation

*iscl* (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ares* -- le signal de sortie au taux audio.

*asig* -- le signal d'entrée au taux audio.

*kcf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*kbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*areson* est un filtre dont les fonctions de transfert sont complémentaires de celles de *reson*. Ainsi *areson* est un filtre réjecteur de bande variable (notch filter) dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *areson* mais reste le complément réel de l'unité correspondante. Ainsi les deux versions d'un signal audio filtré par des unités *reson* et *areson* correspondantes, redonneraient par addition le signal original.

Cette propriété est particulièrement utile pour contrôler le mélange de différentes sources (voir *lpreson*). On peut obtenir des courbes de réponse complexes comme celles qui présentent plusieurs valeurs maximales, en utilisant une banque de filtres adéquats en série. (La réponse résultante est le produit des différentes réponses.) Dans une telle situation, les atténuations combinées peuvent conduire à une sérieuse perte de puissance du signal, mais celle-ci peut être restaurée au moyen de *balance*.

## Exemples

Voici un exemple de l'opcode *areson*. Il utilise le fichier *areson.csd* [examples/areson.csd].

### Exemple 44. Exemple de l'opcode *areson*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d           ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o areson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
; Generate a white noise signal.
asig rand 20000

out asig
endin

; Instrument #2 - a filtered noise waveform.
instr 2
; Generate a white noise signal.
asig rand 20000

; Filter it using the areson opcode.
kcf init 1000
kbw init 100
afilt areson asig, kcf, kbw

; Clip the filtered signal's amplitude to 85 dB.
a1 clip afilt, 2, ampdb(85)
out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsSoundSynthesizer>
```

## Voir Aussi

*aresonk, atone, atonek, portk, reson, resonk, tone, tonek*

# aresonk

**aresonk** — Un filtre réjecteur de bande réglable (notch filter) dont les fonctions de transfert sont les complémentaires de celles de l'opcode *reson*.

## Description

Un filtre réjecteur de bande réglable dont les fonctions de transfert sont les complémentaires de celles de l'opcode *reson*.

## Syntaxe

```
kres aresonk ksig, kcf, kbw [, iscl] [, iskip]
```

## Initialisation

*iscl* (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*kcf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*kbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*aresonk* est un filtre dont les fonctions de transfert sont complémentaires de celles de *resonk*. Ainsi *aresonk* est un filtre réjecteur de bande variable (notch filter) dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *aresonk* mais reste le complément réel de l'unité correspondante.

## Voir aussi

*areson*, *atone*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tone*, *tonek*



# atone

*atone* — Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tone*.

## Description

Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tone*.

## Syntaxe

```
ares atone asig, khp [, iskip]
```

## Initialisation

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ares* -- le signal de sortie au taux audio.

*asig* -- le signal d'entrée au taux audio.

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

*atone* est un filtre dont les fonctions de transfert sont complémentaires de celles de *tone*. Ainsi *atone* est un filtre passe-haut dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *atone* mais reste le complément réel de l'unité correspondante. Ainsi les deux versions d'un signal audio filtré par des unités *tone* et *atone* correspondantes, redonneraient par addition le signal original.

Cette propriété est particulièrement utile pour contrôler le mélange de différentes sources (voir *lpreson*). On peut obtenir des courbes de réponse complexes comme celles qui présentent plusieurs valeurs maximales, en utilisant une banque de filtres adéquats en série. (La réponse résultante est le produit des différentes réponses.) Dans une telle situation, les atténuations combinées peuvent conduire à une sérieuse perte de puissance du signal, mais celle-ci peut être restaurée au moyen de *balance*.

## Exemples

Voici un exemple de l'opcode *atone*. Il utilise le fichier *atone.csd* [examples/atone.csd].

### Exemple 45. Exemple de l'opcode *atone*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc          -d          ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o atone.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
; Generate a white noise signal.
asig rand 20000

out asig
endin

; Instrument #2 - a filtered noise waveform.
instr 2
; Generate a white noise signal.
asig rand 20000

; Filter it using the atone opcode.
khp init 2000
afilt atone asig, khp

; Clip the filtered signal's amplitude to 85 dB.
al clip afilt, 2, ampdb(85)
out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*areson, aresonk, atonek, port, portk, reson, resonk, tone, tonek*

# atonek

*atonek* — Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tonek*.

## Description

Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tonek*.

## Syntaxe

```
kres atonek ksig, khp [, iskip]
```

## Initialisation

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

*atonek* est un filtre dont les fonctions de transfert sont complémentaires de celles de *tonek*. Ainsi *atonek* est un filtre passe-haut dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *atonek* mais reste le complément réel de l'unité correspondante.

## Voir Aussi

*areson*, *aresonk*, *atone*, *port*, *portk*, *reson*, *resonk*, *tone*, *tonek*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# atonex

atonex — Emule une série de filtres utilisant l'opcode *atone*.

## Description

*atonex* est équivalent à un filtre constitué de plusieurs couches de filtres *atone* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupure plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de k cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

## Syntaxe

```
ares atonex asig, khp [, inumlayer] [, iskip]
```

## Initialisation

*inumlayer* (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

## Voir Aussi

*resonx*, *tonex*

## Crédits

Auteur : Gabriel Maldonado (adapté par John ffitich)  
Italie

Nouveau dans la version 3.49 de Csound

# atrirand

atrirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.

# ATSadd

ATSadd — utilise les données d'un fichier d'analyse ATS pour réaliser une synthèse additive.

## Description

*ATSadd* lit depuis un fichier d'analyse ATS et utilise les données pour réaliser une synthèse additive à partir d'une batterie interne d'oscillateurs avec interpolation.

## Syntaxe

```
ar ATSadd ktimepnt, kfmod, iatsfile, ifn, ipartials[, ipartialoffset, \
    ipartialincr, igatefn]
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ifn* – numéro de table d'une fonction stockée contenant une onde sinus pour *ATSadd* et une onde cosinus pour *ATSaddnz* (voir les exemples ci-dessous pour plus d'information).

*ipartials* – nombre de partiels qui seront utilisés dans la resynthèse (le bruit a un maximum de 25 bandes).

*ipartialoffset* (facultatif) – le premier partiel utilisé (0 par défaut).

*ipartialincr* (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

*igatefn* (facultatif) – numéro d'une fonction stockée qui sera appliquée aux amplitudes des bins de l'analyse avant la resynthèse. Si *igatefn* est supérieur à 0 les amplitudes de chaque bin seront pondérées par *igatefn* selon un simple procédé de mise en correspondance. D'abord les amplitudes de tous les bins de toutes les trames du fichier d'analyse sont comparées pour déterminer la valeur maximale de l'amplitude. Cette valeur est ensuite utilisée pour créer des amplitudes normalisées comme indices dans la fonction stockée *igatefn*. L'amplitude maximale correspondra au dernier point de la fonction. Une amplitude de 0 correspondra au premier point de la fonction. Les valeurs comprises entre 0 et 1 correspondront aux points à l'intérieur de la table de fonction. Voir les exemples ci-dessous.

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSadd* exactement de la même manière que pour *pvoc*.

*ATSadd* et *ATSaddnz* sont basés sur *pvadd* par Richard Karpen et ils utilisent des fichier créés par *ATS* de Juan Pampin (*Analyse* - *Transformation* - *Synthèse* [<http://www-ccrma.stanford.edu/~juan/ATS.html>]).

*kfmod* – Un facteur de transposition du taux de contrôle : la valeur 1 implique pas de transposition, 1.5 transpose vers l'aigu d'une quinte juste et 0.5 vers le grave d'une octave. Est utilisé pour *ATSadd* exactement de la même manière que pour *pvoc*.

*ATSadd* lit depuis un fichier d'analyse ATS et utilise les données pour réaliser une synthèse additive à

partir d'une batterie interne d'oscillateurs avec interpolation. L'utilisateur fournit la table d'onde (habituellement une période d'onde sinusoïdale) et il peut choisir quels partiels de l'analyse seront utilisés dans la resynthèse.

## Exemples

```
ktime line 0, p3, 2.5
asig ATSadd ktime, 1, "clarinet.ats", 1, 20, 2
```

Dans l'exemple ci-dessus, *ipartials* vaut 20 et *ipartialoffset* vaut 2. Les partiels du fichier d'analyse "clarinet.ats" allant du 3ème au 22ème seront synthétisés. *kfmod* vaut 1 et il n'y aura ainsi pas de modification de la hauteur. Comme l'enveloppe *ktimepnt* évolue de 0 à 2.5 pendant la durée de la note, le fichier d'analyse sera lu de 0 à 2.5 secondes de la durée originale de l'analyse pendant la durée de la note csound, ce qui permet de changer la durée indépendamment de la hauteur.

```
ktime line 0, p3, 2.5
asig ATSadd ktime, 1.0125, "clarinet.ats", 1, 20, 0, 2
```

Dans l'exemple ci-dessus nous synthétisons 20 partiels comme dans l'exemple 1 sauf que cette fois-ci nous fixons *ipartialoffset* à 0 et *ipartialincr* à 2, ce qui veut dire que nous commençons avec le premier partiel et que nous synthétisons au total 20 partiels, ignorant tous les partiels impairs (1, 3, 5, ...). Nous élevons aussi la hauteur du résultat (*kfmod* vaut 1.0125).

## Voir Aussi

*ATSread*, *ATSreadnz*, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSaddnz*, *ATSsinnoi*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSaddnz

ATSaddnz — utilise les données d'un fichier d'analyse ATS pour réaliser une synthèse de bruit.

## Description

*ATSaddnz* lit depuis un fichier d'analyse ATS et utilise les données pour réaliser une synthèse additive en utilisant une fonction randi modifiée.

## Syntaxe

```
ar ATSaddnz ktimepnt, iatsfile, ibands[, ibandoffset, ibandincr]
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ibands* – nombre de bandes de bruit qui seront utilisées dans la resynthèse (le bruit comprend 25 bandes au maximum).

*ibandoffset* (facultatif) – la première bande de bruit utilisée (0 par défaut).

*ibandincr* (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ibandoffset* dans la resynthèse (1 par défaut).

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSaddnz* exactement de la même manière que pour *pvoc* et *ATSadd*.

*ATSaddnz* et *ATSadd* sont basés sur *pvadd* par Richard Karpen et ils utilisent des fichier créés par ATS de Juan Pampin (*Analyse* - *Transformation* - *Synthèse* [<http://www-ccrma.stanford.edu/~juan/ATS.html>]).

*ATSaddnz* lit aussi depuis un fichier d'analyse ATS mais il resynthétise le bruit depuis les données d'énergie du bruit contenues dans le fichier ATS. Il utilise une fonction randi modifiée pour créer du bruit à bande limitée et le module avec une onde cosinus, pour synthétiser une sélection de bandes de fréquence spécifiée par l'utilisateur. La modulation du bruit est nécessaire pour placer le bruit à bande limitée au bon endroit dans le spectre de fréquence.

## Exemples

```
ktime line 0, p3, 2.5  
asig ATSaddnz ktime, "clarinet.ats", 25
```

Dans l'exemple ci-dessus nous resynthétisons les 25 bandes de bruit depuis les données contenues dans le fichier d'analyse nommé "clarinet.ats".

```
ktime line 2.5, p3, 0  
asig ATSaddnz ktime, 1, "clarinet.ats", 1, 24
```



Ici nous ne resynthétisons que la 25ème bande de bruit (*ibandoffset* à 24 et *ibands* à 1). De plus notre pointeur de temps évolue de 2.5 à 0 pendant la durée de la note ce qui fait que nous lisons le fichier d'analyse à l'envers à partir de 2.5 secondes.

## Voir Aussi

*ATSread*, *ATSreadnz*, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSaddnz*, *ATSsinnoi*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSbufread

*ATSbufread* — lit des données depuis un fichier ATS et les stocke dans une table interne de paires de données fréquence, amplitude.

## Description

*ATSbufread* lit des données depuis un fichier ATS et les stocke dans une table interne de paires de données fréquence, amplitude.

## Syntaxe

```
ATSbufread ktimepnt, kfmod, iatsfile, ipartials[, ipartialoffset, \
ipartialincr]
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ipartials* – nombre de partiels qui seront utilisés dans la resynthèse (le bruit a un maximum de 25 bandes).

*ipartialoffset* (facultatif) – le premier partiel utilisé (0 par défaut).

*ipartialincr* (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSbufread* exactement de la même manière que pour *pvoc*.

*kfmod* – une entrée pour faire une transposition de hauteur ou une modulation de fréquence sur tous les partiels synthétisés ; si aucune modulation de fréquence ou aucun changement de hauteur ne sont désirés, il faut utiliser 1 pour cette valeur.

*ATSbufread* est basé sur *pvbufread* par Richard Karpen. *ATScross*, *ATSinterpread* et *ATSpartialtap* dépendent tous de *ATSbufread* comme *pvcross* et *pvinterp* dépendent de *pvbufread*. *ATSbufread* lit des données depuis un fichier ATS et les stocke dans une table interne de paires de données fréquence, amplitude. Les données stockées par un *ATSbufread* ne sont accessibles que par d'autres générateurs unitaires, et ainsi, à cause de l'architecture de Csound, un *ATSbufread* doit se trouver avant (mais pas nécessairement directement) tout générateur unitaire dépendant. Bien que *ATSbufread* ne produise pas de données directement, il fonctionne exactement comme *ATSadd*. Il utilise un pointeur temporel (*ktimepnt*) pour indexer les données dans la durée, *ipartials*, *ipartialoffset* et *ipartialincr* pour sélectionner les partiels à stocker dans la table et *kfmod* pour pondérer les partiels en fréquence.

## Exemples

Voir les exemples de *ATScross*, *ATSinterpread* et *ATSpartialtap*

## Voir Aussi

*ATSread, ATSreadnz, ATSinfo, ATSinnoi, ATScross, ATSinterpread, ATSpartialtap, ATSadd, ATSaddnz*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATScross

ATScross — exécute une synthèse croisée à partir de fichiers d'analyse ATS.

## Description

*ATScross* utilise les données d'un fichier d'analyse ATS et d'un *ATSbufread* pour exécuter une synthèse croisée.

## Syntaxe

```
ar ATScross ktimepnt, kfmod, iatsfile, ifn, kmylev, kbuflev, ipartials \  
    [, ipartialoffset, ipartialincr]
```

## Initialisation

*iatsfile* – entier ou chaîne de caractères dénotant un fichier de contrôle dérivé de l'analyse ATS d'un signal audio. Un entier indique le suffixe d'un fichier ATS.m ; une chaîne de caractères (entre guillemets) donne un nom de fichier, ou un nom de chemin complet. Si ce n'est pas un chemin complet, le fichier est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SADIR (si elle est définie).

*ifn* – numéro de la table d'une fonction stockée contenant une onde sinusoïdale.

*ipartials* – nombre de partiels qui seront utilisés dans la resynthèse.

*ipartialoffset* (facultatif) – le premier partiel utilisé (0 par défaut).

*ipartialincr* (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATScross* exactement de la même manière que pour *pvoc*.

*kfmod* – une entrée pour faire une transposition de hauteur ou une modulation de fréquence sur tous les partiels synthétisés ; si aucune modulation de fréquence ou aucun changement de hauteur ne sont désirés, il faut utiliser 1 pour cette valeur.

*kmylev* - pondère le composant *ATScross* du spectre de fréquence appliqué aux partiels depuis le fichier ATS indiqué par l'opcode *ATScross*. L'information du spectre de fréquence vient du fichier ATS de *ATScross*. Une valeur de 1 (et 0 pour *kbuflev*) donne le même résultat que *ATSadd*.

*kbuflev* - pondère le composant *ATSbufread* du spectre de fréquence appliqué aux partiels depuis le fichier ATS indiqué par l'opcode *ATScross*. L'information du spectre de fréquence vient du fichier ATS *ATSbufread*. Une valeur de 1 (et 0 pour *kmylev*) donne des partiels qui ont l'information de fréquence du fichier ATS donné par l'*ATScross*, mais les amplitudes imposées par les données du fichier ATS donné par *ATSbufread*.

*ATScross* utilise les données d'un fichier d'analyse ATS (indiqué par *iatsfile*) et les données d'un *ATSbufread* pour exécuter une synthèse croisée. *ATScross* utilise *ktimepnt*, *kfmod*, *ipartials*, *ipartialoffset* et *ipartialincr* de la même manière que *ATSadd*. *ATScross* synthétise une onde sinus pour chaque partiel

sélectionné par l'utilisateur et utilise la fréquence de ce partiel (après pondération en fréquence par *kfmod*) comme indice dans la table créée par *ATShufread*. Les valeurs intermédiaires sont obtenues par interpolation. *ATScross* utilise la somme des données d'amplitude de son fichier ATS (pondérée par *kmylev*) et les données d'amplitude fournies par *ATShufread* (pondérées par *kbuflev*) pour mettre à l'échelle l'amplitude de chaque partiel qu'il synthétise. En fixant *kmylev* à un et *kbuflev* à zéro, *ATScross* agira exactement comme *ATSadd*. En fixant *kmylev* à zéro et *kbuflev* à un, on produira un son qui aura tous les partiels sélectionnés par l'unité *ATScross*, mais avec les amplitudes fournies par *ATShufread*. Il n'est pas nécessaire que le pointeur de temps de l'*ATShufread* soit le même que celui de l'*ATScross*.

## Exemples

```

ktime  line      0, p3, 2.4
ktime2 line      0, p3, 0.5
kline  expseg    0.001, 0.9, 1, p3-0.9, 1
kline2 expseg    0.001, p3, 1
        ATShufread ktime2, 1, "crt.ats", 20
aout   ATScross  ktime, 1, "cl.ats", 1, kline, 0.001 * (1 - kline2), 42

```

Cet exemple exécute une synthèse croisée à partir des deux fichiers ATS "crt.ats" et "cl.ats". Le résultat sera un son qui débute avec le profil (en fréquence) de crt.ats et se termine avec le profil de cl.ats. Toutes les fréquences d'onde sinusoïdale viennent de cl.ats. La valeur de *kbuflev* est pondérée parce que l'énergie produite en appliquant le spectre de fréquence de crt.ats aux partiels de cl.ats est très importante. Noter également que les pointeurs de temps d'*ATShufread* et d'*ATScross* n'ont pas nécessairement la même valeur, ce qui permet de lire deux fichiers ATS à des vitesses différentes.

## Voir Aussi

*ATSread*, *ATSreadnz*, *ATSinfo*, *ATSinnoi*, *ATShufread*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*

## Crédits

Auteur : Alex Norman  
 Seattle, Washington  
 2004

# ATSinfo

ATSinfo — lit des données de l'en-tête d'un fichier ATS.

## Description

*atsinfo* lit des données de l'en-tête d'un fichier ATS.

## Syntaxe

```
idata ATSinfo iatsfile, ilocation
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ilocation* – indique quel champ de l'en-tête du fichier retourner. Les données de l'en-tête donnent de l'information sur les données contenues dans le reste du fichier ATS. Les valeurs possibles pour *ilocation* sont données dans la liste suivante :

0 - Taux d'échantillonnage (Hz)

1 - Taille de trame (en échantillons)

2 - Taille de fenêtre (en échantillons)

3 - Nombre de partiels

4 - Nombre de trames

5 - Amplitude maximale

6 - Fréquence maximale (Hz)

7 - Durée (secondes)

8 - Type du fichier ATS

## Exécution

Des macros peuvent améliorer la lisibilité de votre code Csound ; je donne mes définitions de macro ci-dessous :

```
#define ATS_SAMP_RATE #0#  
#define ATS_FRAME_SZ #1#  
#define ATS_WIN_SZ #2#  
#define ATS_N_PARTIALS #3#  
#define ATS_N_FRAMES #4#  
#define ATS_AMP_MAX #5#  
#define ATS_FREQ_MAX #6#  
#define ATS_DUR #7#  
#define ATS_TYPE #8#
```

*ATSinfo* peut être utile pour écrire des instruments génériques qui fonctionneront avec plusieurs fichiers

ATS, même s'ils ont différentes longueurs, différents nombres de partiels, etc. L'exemple 2 est une simple application de cela.

## Exemples

1.

```
imax_freq      ATSinfo "cl.ats", $ATS_FREQ_MAX
```

Dans l'exemple ci-dessus nous obtenons la valeur de la fréquence maximale du fichier ATS "cl.ats" et nous la stockons dans `imax_freq`. Nous utilisons la macro Csound `$ATS_FREQ_MAX` (définie ci-dessus), qui est équivalente au nombre 6.

2.

```
i_npartials    ATSinfo p4, $ATS_N_PARTIALS  
i_dur          ATSinfo p4, $ATS_DUR  
ktimepnt       line    0, p3, i_dur  
aout           ATSadd  ktimepnt, 1, p4, 1, i_npartials
```

Dans l'exemple ci-dessus nous utilisons *ATSinfo* pour retrouver la durée et le nombre de partiels dans le fichier ATS indiqué par `p4`. Avec cette information nous synthétisons les partiels au moyen d'*ATSadd*. Comme la durée et le nombre de partiels ne sont pas codés en dur, nous pouvons utiliser ce code avec n'importe quel fichier ATS.

## Voir Aussi

*ATSread*, *ATSreadnz*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*, *ATSsinnoi*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSinterpread

ATSinterpread — permet de déterminer l'enveloppe de fréquence de n'importe quel *ATSbufread*.

## Description

*ATSinterpread* permet de déterminer l'enveloppe de fréquence de n'importe quel *ATSbufread*.

## Syntaxe

```
kamp ATSinterpread kfreq
```

## Exécution

*kfreq* - une valeur de fréquence (en Hz) utilisée par *ATSinterpread* comme indice dans la table produite par un *ATSbufread*.

*ATSinterpread* prend une valeur de fréquence (*kfreq* en Hz). Cette fréquence sert à indexer les données d'un *ATSbufread*. La valeur retournée est une amplitude obtenue de l'*ATSbufread* après interpolation. *ATSinterpread* permet de déterminer l'enveloppe de fréquence de n'importe quel *ATSbufread*. Ces données peuvent être utiles pour plusieurs raisons, dont l'une est la réalisation de la synthèse croisée entre des données provenant d'un fichier ATS et des données non ATS.

## Exemples

```
ptime      line      0, p3, 2.4
           ATSbufread ptime, 1, "cl.ats", 42
kmp         ATSinterpread p4
aosc        oscili      kamp, p4, 1
```

Cet exemple montre comment utiliser *ATSinterpread*. Ici une fréquence est fournie par la partition (p4) et cette fréquence est passée à un *ATSinterpread* (avec un *ATSbufread*) correspondant. L'*ATSinterpread* utilise cette fréquence pour retourner l'amplitude correspondante basée sur le fichier ATS donné par le *ATSbufread* (cl.ats dans ce cas). Nous utilisons ensuite cette amplitude pour pondérer une onde sinus qui est synthétisée avec la même fréquence (p4). On peut étendre ceci pour inclure plusieurs ondes sinus. De cette manière il est possible de synthétiser n'importe quelle fréquence raisonnable (comprise entre la fréquence basse et la fréquence haute du fichier ATS indiqué), et de conserver la forme (en fréquence) du fichier ATS (donné par l'*ATSbufread*).

## Voir Aussi

*ATSread*, *ATSreadnz*, *ATSinfo*, *ATSsinnoi*, *ATSbufread*, *ATScross*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004



# ATSread

ATSread — lit des données depuis un fichier ATS.

## Description

*ATSread* retourne l'information d'amplitude (*kamp*) et de fréquence (*kfreq*) d'un partiel spécifié contenu dans le fichier d'analyse ATS au moment indiqué par le pointeur de temps *ktimepnt*.

## Syntaxe

```
kfreq, kamp ATSread ktimepnt, iatsfile, ipartial
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ipartial* – le numéro du partiel d'analyse duquel seront retournés la fréquence en Hz et l'amplitude.

## Exécution

*kfreq*, *kamp* - sorties de l'unité *ATSread*. Ces valeurs représentent la fréquence et l'amplitude d'un partiel spécifique sélectionné par *ipartial*. Les informations du partiel sont dérivées d'une analyse ATS. *ATSread* interpole la fréquence et l'amplitude entre les trames dans le fichier d'analyse ATS au taux-k. La sortie dépend des données dans le fichier d'analyse et du pointeur *ktimepnt*.

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSread* exactement de la même manière que pour *pvoc* et *ATSadd*.

## Exemples

```
ktime      line      0, p3, 2.5  
kfreq, kamp ATSread ktime, "clarinet.ats", 2  
aout       oscili    1000000 * kamp, kfreq, 1
```

Ici nous utilisons *ATSread* pour obtenir la fréquence et l'amplitude du second partiel du fichier d'analyse ATS 'clarinet.ats'. Nous utilisons ces données pour piloter un oscillateur, mais nous pourrions les utiliser pour toute autre opération qui accepte une entrée au taux-k, comme la largeur de bande et la résonnance d'un filtre, etc.

## Voir Aussi

*ATSreadnz*, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpretad*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*, *ATSsinnoi*

## Crédits

Auteur : Alex Norman

Seattle, Washington  
2004

# ATSreadnz

ATSreadnz — lit des données depuis un fichier ATS.

## Description

*ATSreadnz* retourne l'énergie (*kenergy*) d'une bande de bruit spécifiée par l'utilisateur (1-25 bandes) à la date indiquée par le pointeur de temps *ktimepnt*.

## Syntaxe

```
kenergy ATSreadnz ktimepnt, iatsfile, iband
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*iband* – le numéro de la bande de bruit dont il faut retourner les données d'énergie.

## Exécution

*kenergy* est la sortie contenant l'énergie interpolée linéairement de la bande de bruit indiquée par *iband*. La sortie dépend des données dans le fichier d'analyse et de *ktimepnt*.

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSreadnz* exactement de la même manière que pour *pvoc* et *ATSadd*.

*ATSaddnz* lit depuis un fichier ATS et resynthétise le bruit à partir des données d'énergie du bruit contenues dans le fichier ATS. Il utilise une fonction *randi* modifiée pour créer du bruit à bande limitée et le module avec une table d'onde fournie par l'utilisateur (une période d'une onde cosinus), pour synthétiser une sélection spécifiée par l'utilisateur de bandes de fréquence. Il est nécessaire de moduler le bruit pour placer le bruit à bande limitée au bon endroit dans le spectre de fréquence.

Une analyse ATS diffère d'une analyse *pval* du fait qu'ATS trace les partiels et calcule l'énergie du bruit dans le son étant analysé. Pour plus d'information sur l'analyse ATS, lire la description de Juan Pampin sur la page web *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

## Exemples

```
ktime   line      2.5, p3, 0  
kenergy ATSreadnz ktime, "clarinet.ats", 5
```

Ici nous extrayons la bande d'énergie 5 du bruit du fichier d'analyse ATS 'clarinet.ats'. Nous lisons à l'envers depuis 2.5 secondes vers le début du fichier d'analyse. On peut l'utiliser pour synthétiser du bruit comme cela :

```
anoise  randi      sqrt(kenergy), 55  
aout    oscili     40000000000000000000000000000000, 455, 2  
aout    =           aout * anoise
```

La table de fonction 2 utilisée dans l'oscillateur est une onde cosinus, qui est nécessaire pour déplacer le bruit à bande limitée au bon endroit dans le spectre de fréquence. La fonction *randi* crée une bande de fréquence centrée sur 0 Hz qui a une largeur de bande d'environ 110 Hz ; en la multipliant par une onde cosinus on la déplace pour qu'elle soit centrée à 455 Hz, qui est la fréquence centrale de la 5ème bande critique de bruit. Ce n'est qu'un exemple pour synthétiser du bruit qui serait mieux réalisé avec *ATSaddnz*, à moins que l'on ne désire utiliser son propre algorithme de synthèse de bruit. Peut-être peut-on utiliser l'énergie du bruit pour autre chose comme appliquer une petite quantité de tremblement à des partiels spécifiques ou contrôler quelque chose sans aucun rapport avec le son source ?

## Voir Aussi

*ATSread*, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*, *ATSSinnoi*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSpartialtap

ATSpartialtap — retourne une paire fréquence, amplitude à partir d'un opcode *ATSbufread*.

## Description

*ATSpartialtap* prend un numéro de partiel et retourne une paire fréquence, amplitude. Les données de fréquence et d'amplitude proviennent d'un opcode *ATSbufread*.

## Syntaxe

```
kfrq, kamp ATSpartialtap ipartialnum
```

## Initialisation

*ipartialnum* - indique le partiel que l'opcode *ATSpartialtap* doit lire à partir d'un *ATSbufread*.

## Exécution

*kfrq* - retourne la valeur de fréquence du partiel demandé.

*kamp* - retourne la valeur d'amplitude du partiel demandé.

*ATSpartialtap* prend un numéro de partiel et retourne une paire fréquence, amplitude. Les données de fréquence et d'amplitude proviennent d'un opcode *ATSbufread*. C'est une version restreinte d'*ATSread*, car chaque opcode *ATSread* a son propre pointeur de temps indépendant et *ATSpartialtap* est restreint aux données données par un *ATSbufread*. Cette simplicité est son point fort.

## Exemples

```
ptime      line      0, p3, 2.4
ktime, 1, "crt.ats", 20
kfreq1, kamp1 ATSpartialtap 1
kfreq2, kamp2 ATSpartialtap 10
kfreq3, kamp3 ATSpartialtap 20
```

Cet exemple utilise un *ATSpartialtap* et un *ATSbufread* pour lire les partiels 1, 10 et 20 de 'crt.ats'. On pourrait utiliser ces amplitudes et ces fréquences pour resynthétiser les partiels ou pour faire quelque chose de tout à fait différent.

## Voir Aussi

*ATSread*, *ATSreadnz*, *ATSinfo*, *ATSsinnoi*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSadd*, *ATSaddnz*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSsinnoi

ATSsinnoi — utilise les données d'un fichier d'analyse ATS pour réaliser une resynthèse.

## Description

*ATSsinnoi* lit les données d'un fichier ATS et utilise cette information pour synthétiser à la fois des sinusoïdes et du bruit.

## Syntaxe

```
ar ATSsinnoi ktimepnt, ksinlev, knzlev, kfmod, iatsfile, ipartials \  
    [, ipartialoffset, ipartialincr]
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ipartials* – nombre de partiels qui seront utilisés dans la resynthèse (le bruit a un maximum de 25 bandes).

*ipartialoffset* (optional) – (facultatif) – le premier partiel utilisé (0 par défaut).

*ipartialincr* (optional) – (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSsinnoi* exactement de la même manière que pour *pvoc*.

*ksinlev* - contrôle le niveau des sinus dans le générateur unitaire *ATSsinnoi*. Une valeur de 1 donne des ondes sinus à plein volume.

*knzlev* - contrôle le niveau des composants du bruit dans le générateur unitaire *ATSsinnoi*. Une valeur de 1 donne du bruit à plein volume.

*kfmod* – une entrée pour faire une transposition de hauteur ou une modulation de fréquence sur tous les partiels synthétisés ; si aucune modulation de fréquence ou aucun changement de hauteur ne sont désirés, il faut utiliser 1 pour cette valeur.

*ATSsinnoi* lit les données d'un fichier ATS et utilise cette information pour synthétiser à la fois des sinusoïdes et du bruit. L'énergie du bruit pour chaque bande est distribuée également entre les partiels qui tombent dans cette bande. Chaque partiel est ensuite synthétisé, avec sa composante de bruit. Chaque composante de bruit est ensuite modulée par le partiel correspondant pour être placée au bon endroit dans le spectre de fréquence. Les niveaux du bruit et des partiels sont contrôlables individuellement. Voir la page web *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>] pour plus d'information sur la synthèse sinnoi. Une analyse ATS diffère d'une analyse pvanal du fait qu'ATS trace les partiels et calcule l'énergie du bruit dans le son étant analysé. Pour plus d'information sur l'analyse ATS, lire la description de Juan Pampin sur la page web *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

## Exemples

```
ptime  line 0, p3, 2.5
asig  ATSSinnoi ktime, 1, 1, 1, "clarinet.ats", 42
```

Nous synthétisons ici à la fois le bruit et les ondes sinus (les 42 partiels) contenus dans "clarinet.ats". Les volumes relatifs du bruit et des partiels sont inchangés (chacun est fixé à 1).

```
ptime  line 0, p3, 2.5
knzfade expon 0.001, p3, 2.5
asig  ATSSinnoi ktime, 1, knzfade, 1, "clarinet.ats", 42
```

Cet exemple reprend le précédent mais en utilisant une enveloppe pour contrôler *knzlev* (le niveau de bruit). Cela donne un son de clarinette dont la composante de bruit apparaît progressivement durant la durée de la note.

## Voir Aussi

*ATSread*, *ATSreadnz*, *ATSinio*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# aunirand

aunirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.



# aweibull

aweibull — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

# babo

babo — A physical model reverberator.

## Description

*babo* stands for *ball-within-the-box*. It is a physical model reverberator based on the paper by Davide Rocchesso "The Ball within the Box: a sound-processing metaphor", Computer Music Journal, Vol 19, N.4, pp.45-47, Winter 1995.

The resonator geometry can be defined, along with some response characteristics, the position of the listener within the resonator, and the position of the sound source.

## Syntax

```
a1, a2 babo asig, ksrcx, ksrcy, ksrcz, irx, iry, irz [, idiff] [, ifno]
```

## Initialization

*irx, iry, irz* -- the coordinates of the geometry of the resonator (length of the edges in meters)

*idiff* -- is the coefficient of diffusion at the walls, which regulates the amount of diffusion (0-1, where 0 = no diffusion, 1 = maximum diffusion - default: 1)

*ifno* -- expert values function: a function number that holds all the additional parameters of the resonator. This is typically a GEN2--type function used in non-rescaling mode. They are as follows:

- *decay* -- main decay of the resonator (default: 0.99)
- *hydecay* -- high frequency decay of the resonator (default: 0.1)
- *rcvx, rcvy, rcvz* -- the coordinates of the position of the receiver (the listener) (in meters; 0,0,0 is the resonator center)
- *rdistance* -- the distance in meters between the two pickups (your ears, for example - default: 0.3)
- *direct* -- the attenuation of the direct signal (0-1, default: 0.5)
- *early\_diff* -- the attenuation coefficient of the early reflections (0-1, default: 0.8)

## Performance

*asig* -- the input signal

*ksrcx, ksrcy, ksrcz* -- the virtual coordinates of the source of sound (the input signal). These are allowed to move at k-rate and provide all the necessary variations in terms of response of the resonator.

## Examples

Here is a simple example of the babo opcode. It uses the file *babo.csd* [examples/babo.csd], and *beats.wav* [examples/beats.wav].

**Exemple 46. A simple example of the babo opcode.**

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o babo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Nicola Bernardini */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; minimal babo instrument
;
instr 1
    ix      = p4 ; x position of source
    iy      = p5 ; y position of source
    iz      = p6 ; z position of source
    ixsize  = p7 ; width of the resonator
    iysize  = p8 ; depth of the resonator
    izsize  = p9 ; height of the resonator

    ainput soundin "beats.wav"

    al,ar babo    ainput*0.7, ix, iy, iz, ixsize, iysize, izsize

    outs    al,ar

endin

</CsInstruments>
<CsScore>

/* Written by Nicola Bernardini */
; simple babo usage:
;
;p4      : x position of source
;p5      : y position of source
;p6      : z position of source
;p7      : width of the resonator
;p8      : depth of the resonator
;p9      : height of the resonator
;
i 1 0 10 6 4 3 14.39 11.86 10
;          ^^^^^^  ^^^^^^^^^^^^^^^^^
;          |||/|||/  ++++++: optimal room dims according to
;          |||/|||/  ++++++ Milner and Bernard JASA 85(2), 1989
;          ++++++: source position
e

</CsScore>
</CsoundSynthesizer>
```

Here is an advanced example of the babo opcode. It uses the file *babo\_expert.csd* [examples/babo\_expert.csd], and *beats.wav* [examples/beats.wav].

**Exemple 47. An advanced example of the babo opcode.**

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o babo_expert.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Nicola Bernardini */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; full blown babo instrument with movement
;
instr 2
ixstart = p4 ; start x position of source (left-right)
ixend = p7 ; end x position of source
iystart = p5 ; start y position of source (front-back)
iyend = p8 ; end y position of source
izstart = p6 ; start z position of source (up-down)
izend = p9 ; end z position of source
ixsize = p10 ; width of the resonator
iysize = p11 ; depth of the resonator
izsize = p12 ; height of the resonator
idiff = p13 ; diffusion coefficient
iexpert = p14 ; power user values stored in this function

ainput soundin "beats.wav"
ksource_x line ixstart, p3, ixend
ksource_y line iystart, p3, iyend
ksource_z line izstart, p3, izend

al,ar babo ainput*0.7, ksource_x, ksource_y, ksource_z, ixsize, iysize, izsize, idiff, iexpert

outs al,ar

endin

</CsInstruments>
<CsScore>

/* Written by Nicola Bernardini */
; full blown instrument
;p4 : start x position of source (left-right)
;p5 : end x position of source
;p6 : start y position of source (front-back)
;p7 : end y position of source
;p8 : start z position of source (up-down)
;p9 : end z position of source
;p10 : width of the resonator
;p11 : depth of the resonator
;p12 : height of the resonator
;p13 : diffusion coefficient
;p14 : power user values stored in this function

; decay hidecay rx ry rz rdistance direct early_diff
f1 0 8 -2 0.95 0.95 0 0 0 0.3 0.5 0.8 ; brighter
f2 0 8 -2 0.95 0.5 0 0 0 0.3 0.5 0.8 ; default (to be set as)
f3 0 8 -2 0.95 0.01 0 0 0 0.3 0.5 0.8 ; darker
f4 0 8 -2 0.95 0.7 0 0 0 0.3 0.1 0.4 ; to hear the effect of diffusion
f5 0 8 -2 0.9 0.5 0 0 0 0.3 2.0 0.98 ; to hear the movement
f6 0 8 -2 0.99 0.1 0 0 0 0.3 0.5 0.8 ; default vals
;
; ----- gen. number: negative to avoid rescaling

i2 0 10 6 4 3 6 4 3 14.39 11.86 10 1 6 ; defaults
i2 + 4 6 4 3 6 4 3 14.39 11.86 10 1 1 ; hear brightness 1
i2 + 4 6 4 3 -6 -4 3 14.39 11.86 10 1 2 ; hear brightness 2
i2 + 4 6 4 3 -6 -4 3 14.39 11.86 10 1 3 ; hear brightness 3
i2 + 3 .6 .4 .3 -.6 -.4 .3 1.439 1.186 1.0 0.0 4 ; hear diffusion 1
i2 + 3 .6 .4 .3 -.6 -.4 .3 1.439 1.186 1.0 1.0 4 ; hear diffusion 2
i2 + 4 12 4 3 -12 -4 -3 24.39 21.86 20 1 5 ; hear movement
;

```

```

i2 + 4 6 4 3 6 4 3 14.39 11.86 10 1 1 ; hear brightness 1
i2 + 4 6 4 3 -6 -4 3 14.39 11.86 10 1 2 ; hear brightness 2
i2 + 4 6 4 3 -6 -4 3 14.39 11.86 10 1 3 ; hear brightness 3
i2 + 3 .6 .4 .3 -.6 -.4 .3 1.439 1.186 1.0 0.0 4 ; hear diffusion 1
i2 + 3 .6 .4 .3 -.6 -.4 .3 1.439 1.186 1.0 1.0 4 ; hear diffusion 2
i2 + 4 12 4 3 -12 -4 -3 24.39 21.86 20 1 5 ; hear movement
;
; /////////////////////////////////////////////////// | --: expert values function
; /////////////////////////////////////////////////// +--: diffusion
; /////////////////////////////////////////////////// -----: optimal room dims according to Milner and Bernard JASA 8
; ///////////////////////////////////////////////////
; -----: source position start and end
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Paolo Filippi  
Padova, Italy  
1999

Nicola Bernardini  
Rome, Italy  
2000

New in Csound version 4.09

# balance

balance — Ajuste un signal audio selon les valeurs d'un autre.

## Description

La valeur efficace de *asig* peut être interrogée, fixée ou ajustée pour s'adapter à celle d'un signal de comparaison.

## Syntaxe

```
ares balance asig, acomp [, ihp] [, iskip]
```

## Initialisation

*ihp* (facultatif) -- point à mi-puissance (en Hz) d'un d'un filtre passe-bas interne spécial. La valeur par défaut est 10.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace de données interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio en entrée

*acomp* -- le signal de comparaison

*balance* restitue une version de *asig*, dont l'amplitude a été modifiée de façon à ce que sa valeur efficace soit égale à celle d'un signal de comparaison *acomp*. Ainsi un signal qui a subi une perte de puissance (par exemple en traversant un banc de filtres) peut être restauré en l'ajustant, par exemple, à sa propre source. Il faut noter que *gain* et *balance* n'effectuent que des modifications d'amplitude, les signaux de sortie ne subissant aucune autre altération.

## Exemples

Voici un exemple de l'opcode *balance*. Il utilise le fichier *balance.csd* [examples/balance.csd].

### Exemple 48. Exemple de l'opcode *balance*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o balance.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a band-limited pulse train.
asrc buzz 30000, 440, sr/440, 1

; Send the source signal through 2 filters.
a1 reson asrc, 1000, 100
a2 reson a1, 3000, 500

; Balance the filtered signal with the source.
afin balance a2, asrc

out afin
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*gain, rms*

# bamboo

bamboo — Modèle semi-physique d'un son de bambou.

## Description

*bamboo* est un modèle semi-physique d'un son de bambou. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares bamboo kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \  
      [, ifreq1] [, ifreq2]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 1,25.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$\text{damping\_amount} = 0,9999 + (\text{idamp} * 0,002)$

La valeur par défaut de *damping\_amount* est 0,9999 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,05.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2800.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 2240.

*ifreq2* (facultatif) -- La seconde fréquence de résonance. La valeur par défaut est 3360.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode bamboo. Il utilise le fichier *bamboo.csd* [examples/bamboo.csd].

### Exemple 49. Exemple de l'opcode bamboo.



Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o bamboo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 01 ;example of bamboo
al  bamboo p4, 0.01
    out a1
endin

</CsInstruments>
<CsScore>

i1 0 1 20000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*dripwater, guiro, sleighbells, tambourine*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitch

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# barmodel

barmodel — Crée un timbre similaire à une barre de métal frappée.

## Description

La sortie audio est un timbre semblable à celui d'une barre de métal frappée, mettant en œuvre un modèle physique développé à partir de la résolution de l'équation différentielle. On contrôle les conditions aux limites ainsi que les caractéristiques de la barre.

## Syntaxe

```
ares barmodel kbcL, kbcR, iK, ib, kscan, iT30, ipos, ivel, iwid
```

## Initialisation

*iK* -- paramètre de raideur sans dimension. Si ce paramètre est négatif, l'initialisation est ignorée et l'état précédent de la barre est prolongé.

*ib* -- paramètre de perte des hautes fréquences (à garder petit).

*iT30* -- temps de décroissance à 30 db en secondes.

*ipos* -- position le long de la barre où a lieu la frappe.

*ivel* -- vitesse de frappe normalisée.

*iwid* -- largeur spatiale de la frappe.

## Exécution

Une note est jouée sur une barre métallique, avec les arguments suivants.

*kbcL* -- Condition aux limites à l'extrémité gauche de la barre (1 fixée, 2 pivotante, 3 libre).

*kbcR* -- Condition aux limites à l'extrémité droite de la barre (1 fixée, 2 pivotante, 3 libre).

*kscan* -- Taux de lecture de la position de sortie.

Noter que le changement des conditions aux limites pendant l'exécution peut provoquer des bruits parasites ; cette possibilité est offerte à titre expérimental. L'utilisation d'un *kscan* différent de zéro peut produire des réintroductions apparentes du son à cause de la modulation.

## Exemples

Voici un exemple de l'opcode barmodel. Il utilise le fichier *barmodel.csd* [examples/barmodel.csd].

### Exemple 50. Exemple de l'opcode barmodel.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o barmodel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      1

; Instrument #1.
instr 1
  aq      barmodel  1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
  out     aq
endin

</CsInstruments>
<CsScore>

i1 0.0 0.5 3 0.2 500 0.05
i1 0.5 0.5 -3 0.3 1000 0.05
i1 1.0 0.5 -3 0.4 1000 0.1
i1 1.5 4.0 -3 0.5 800 0.05
e
/* barmodel */

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Stefan Bilbao  
 Université d'Edimbourg, UK  
 Auteur : John ffitch  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 5.01 de Csound

# bbcutm

bbcutm — Extrait des segments dans le style breakbeat à partir d'un flux audio mono.

## Description

Le BreakBeat Cutter extrait automatiquement des segments à partir d'un flux audio dans le style des manipulations du "drum and bass/jungle breakbeat". Il y a deux versions, pour les sources mono (*bbcutm*) ou stéréo (*bbcuts*). Bien que basé à l'origine sur les coupures breakbeat, l'opcode peut être appliqué à n'importe quel type de source audio.

La séquence de coupure typique sur une mesure subdivisée en croches serait

$3 + 3R + 2$

dans laquelle nous prenons un bloc de trois unités au début de la source, le répétons, puis deux unités venant des 7èmes et 8èmes croches de la source.

Nous parlons de restituer des phrases (une séquence de coupures avant d'atteindre une nouvelle phrase au début d'une mesure) et des unités (comme subdivisions des notes).

L'opcode donne un rendu plus vivant lorsqu'on utilise simultanément plusieurs versions synchronisées.

## Syntaxe

```
al bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats \  
    [, istutterspeed] [, istutterchance] [, ienvchoice ]
```

## Initialisation

*ibps* -- Tempo pour les coupures, en pulsations par seconde.

*isubdiv* -- Unité de subdivision pour une mesure. Par exemple 8 désigne la croche (dans une mesure à 4/4).

*ibarlength* -- Nombre de pulsations par mesure. Il vaut 4 pour la mesure par défaut à 4/4.

*iphrasebars* -- Les coupures sont générées par phrases, chaque phrase durant *iphrasebars*.

*inumrepeats* -- Dans une utilisation normale, l'algorithme permet une répétition supplémentaire d'une coupure donnée à la fois. Ce paramètre permet de modifier ce comportement. La valeur 1 représente la norme d'une répétition supplémentaire. 0 supprime la répétition et l'on obtient la source originale excepté pour l'enveloppe et le stuttering.

*istutterspeed* -- (facultatif, par défaut=1) Le stutter peut être un multiple entier de la vitesse de subdivision. Par exemple, si *isubdiv* vaut 8 (croches) et *istutterspeed* vaut 2, le stutter est en doubles croches (= subdiv de 16). La valeur par défaut est 1.

*istutterchance* -- (facultatif, par défaut=0) La fin d'une phrase a cette probabilité de devenir l'unité de répétition du stutter (0,0 à 1,0). La valeur par défaut est 0.

*ienvchoice* -- (facultatif, par défaut=1) Choisir 1 pour l'activer (enveloppe exponentielle pour les grains

de coupure) ou 0 pour le désactiver. S'il est désactivé, on entendra des clics, mais ça peut donner de bons résultats bruiteux, en particulier avec les sources percussives. La valeur par défaut est 1, actif.

## Exécution

*asource* -- Le signal sonore à couper. Cette version fonctionne en temps réel sans connaissance des événements audio futurs.

## Exemples

Voici un exemple de l'opcode *bbcutm*. Il utilise les fichiers *bbcutm.csd* [examples/bbcutm.csd] et *beats.wav* [examples/beats.wav].

### Exemple 51. Un exemple simple de l'opcode *bbcutm*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o bbcutm.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - Play an audio file normally.
instr 1
  asource soundin "beats.wav"
  out asource
endin

; Instrument #2 - Cut-up an audio file.
instr 2
  asource soundin "beats.wav"

  ibps = 4
  isubdiv = 8
  ibarlength = 4
  iphrasebars = 1
  inumrepeats = 2

  al bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats

  out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 3 2
e

</CsScore>
</CsoundSynthesizer>
```

Voici quelques exemples plus avancés ...

## Exemple 52. Premiers pas - versions mono et stéréo

```
<CsoundSynthesizer>
<CsInstruments>
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

instr 1
  asource diskin "break7.wav",1,0,1 ; a source breakbeat sample, wraparound lest it stop!

  ; cuts in eighth notes per 4/4 bar, up to 4 bar phrases, up to 1
  ; repeat in total (standard use) rare stuttering at 16 note speed,
  ; no enveloping
  asig bbcutm asource, 2.6937, 8, 4, 4, 1, 2, 0.1, 0

  outs      asig,asig
endin

instr 2 ;stereo version
  asource1,asource2 diskin "break7stereo.wav", 1, 0, 1 ; a source breakbeat sample, wraparound lest it stop!

  ; cuts in eighth notes per 4/4 bar, up to 4 bar phrases, up to 1
  ; repeat in total (standard use) rare stuttering at 16 note speed,
  ; no enveloping
  asig1,asig2 bbcuts asource1, asource2, 2.6937, 8, 4, 4, 1, 2, 0.1, 0

  outs      asig1,asig2
endin

</CsInstruments>
<CsScore>
i1 0 10
i2 11 10
e
</CsScore>
</CsoundSynthesizer>
```

## Exemple 53. Breaks multiples simultanés synchronisés

```
<CsoundSynthesizer>
<CsInstruments>
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

instr 1
  ibps      = 2.6937
  iplaybackspeed = ibps/p5
  asource diskin p4, iplaybackspeed, 0, 1

  asig bbcutm asource, 2.6937, p6, 4, 4, p7, 2, 0.1, 1

  out      asig
endin

</CsInstruments>
<CsScore>

; source      bps cut repeats
i1 0 10 "break1.wav" 2.3 8 2 //2.3 is the source original tempo
i1 0 10 "break2.wav" 2.4 8 3
i1 0 10 "break3.wav" 2.5 16 4
e
```

```
</CsScore>
</CsoundSynthesizer>
```

**Exemple 54. Coupure de n'importe quelle source audio ancienne - des bruits bien plus intéressants que ceux-ci sont possibles !**

```
<CsoundSynthesizer>
<CsInstruments>
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

instr 1
  asource oscil 20000, 70, 1
  ; ain, bps, subdiv, barlength, phrasebars, numrepeats,
  ;stutterspeed, stutterchance, envelopingon
  asig bbcutm asource, 2, 32, 1, 1, 2, 4, 0.6, 1
  outs  asig
endin

</CsInstruments>
<CsScore>
f1 0 256 10 1
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

**Exemple 55. Faux stuttering constant, impossible car on ne peut appliquer le stutter que dans la dernière demie-mesure, pourrait faire un paramètre optionnel supplémentaire de stuterring**

```
<CsoundSynthesizer>
<CsInstruments>
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

instr 1
  asource diskil "break7.wav", 1, 0, 1

  ;16th note cuts- but cut size 2 over half a beat.
  ;each half beat will either survive intact or be turned into
  ;the first sixteenth played twice in succession

  asig bbcutm asource, 2.6937, 2, 0.5, 1, 2, 2, 1.0, 0
  outs  asig
endin

</CsInstruments>
<CsScore>
i1 0 30
e
</CsScore>
</CsoundSynthesizer>
```

**Voir Aussi**

*bbcuts*

## Crédits

Auteur : Nick Collins  
Londres  
Août 2001

Nouveau dans la version 4.13



# bbcuts

bbcuts — Extrait des segments dans le style breakbeat à partir d'un flux audio stéréo.

## Description

Le BreakBeat Cutter extrait automatiquement des segments à partir d'un flux audio dans le style des manipulations du "drum and bass/jungle breakbeat". Il y a deux versions, pour les sources mono (*bbcutm*) ou stéréo (*bbcuts*). Bien que basé à l'origine sur les coupures breakbeat, l'opcode peut être appliqué à n'importe quel type de source audio.

La séquence de coupure typique sur une mesure subdivisée en croches serait

$3 + 3R + 2$

dans laquelle nous prenons un bloc de trois unités au début de la source, le répétons, puis deux unités venant des 7èmes et 8èmes croches de la source.

Nous parlons de restituer des phrases (une séquence de coupures avant d'atteindre une nouvelle phrase au début d'une mesure) et des unités (comme subdivisions des notes).

L'opcode donne un rendu plus vivant lorsqu'on utilise simultanément plusieurs versions synchronisées.

## Syntaxe

```
a1,a2 bbcuts asource1, asource2, ibps, isubdiv, ibarlength, iphrasebars, \
      inumrepeats [, istutterspeed] [, istutterchance] [, ienvchoice]
```

## Initialisation

*ibps* -- Tempo pour les coupures, en pulsations par seconde.

*isubdiv* -- Unité de subdivision pour une mesure. Par exemple 8 désigne la croche (dans une mesure à 4/4).

*ibarlength* -- Nombre de pulsations par mesure. Il vaut 4 pour la mesure par défaut à 4/4.

*iphrasebars* -- Les coupures sont générées par phrases, chaque phrase durant *iphrasebars*.

*inumrepeats* -- Dans une utilisation normale, l'algorithme permet une répétition supplémentaire d'une coupure donnée à la fois. Ce paramètre permet de modifier ce comportement. La valeur 1 représente la norme d'une répétition supplémentaire. 0 supprime la répétition et l'on obtient la source originale excepté pour l'enveloppe et le stuttering.

*istutterspeed* -- (facultatif, par défaut=1) Le stutter peut être un multiple entier de la vitesse de subdivision. Par exemple, si *isubdiv* vaut 8 (croches) et *istutterspeed* vaut 2, le stutter est en doubles croches (= subdiv de 16). La valeur par défaut est 1.

*istutterchance* -- (facultatif, par défaut=0) La fin d'une phrase a cette probabilité de devenir l'unité de répétition du stutter (0,0 à 1,0). La valeur par défaut est 0.

*ienvchoice* -- (facultatif, par défaut=1) Choisir 1 pour l'activer (enveloppe exponentielle pour les grains

de coupure) ou 0 pour le désactiver. S'il est désactivé, on entendra des clics, mais ça peut donner de bons résultats bruiteux, en particulier avec les sources percussives. La valeur par défaut est 1, actif.

## Exécution

*asource* -- Le signal sonore à couper. Cette version fonctionne en temps réel sans connaissance des événements audio futurs.

## Exemples

Voir les exemples avancés dans la notice de l'opcode *bbcutm*.

## Voir Aussi

*bbcutm*

## Crédits

Auteur : Nick Collins  
Londres  
Août 2001

Nouveau dans la version 4.13

# betarand

betarand — Générateur de nombres aléatoires de distribution beta (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution beta (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

ares **betarand** krange, kalpha, kbeta

ires **betarand** krange, kalpha, kbeta

kres **betarand** krange, kalpha, kbeta

## Exécution

*krange* -- l'intervalle des nombres aléatoires (0 - *krange*).

*kalpha* -- valeur de alpha. Si *kalpha* est inférieur à un, ses petites valeurs favorisent les valeurs proches de 0.

*kbeta* -- valeur de beta. Si *kbeta* est inférieur à un, ses petites valeurs favorisent les valeurs proches de *krange*.

Si *kalpha* et *kbeta* sont tous deux égaux à un, nous obtenons une distribution uniforme. Si *kalpha* et *kbeta* sont tous deux supérieurs à un nous obtenons une sorte de distribution gaussienne. Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode betarand. Il utilise le fichier *betarand.csd* [examples/betarand.csd].

### Exemple 56. Exemple de l'opcode betarand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```
-odac          -iadc      ;;;RT audio I/O
; -o betarand.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 441
nchnls = 1

; Instrument #1.
instr 1
; Generate a number between 0 and 1 with a
; uniform distribution.
; krange = 1
; kalpha = 1
; kbeta = 1

; ** every run time same value
il betarand 1, 1, 1

print il
endin

; Instrument #2.
instr 2
; Same of above but with
; seed from system time

seed 0

; ** every run time different value
il betarand 1, 1, 1

print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  il = 24583.412
```

## Voir Aussi

*seed, bexprnd, cauchy, exprand, gauss, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Existait dans la 3.30

Exemple écrit par Kevin Conder.

# bexprnd

bexprnd — Générateur de nombres aléatoires de distribution exponentielle.

## Description

Générateur de nombres aléatoires de distribution exponentielle. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares bexprnd krange
```

```
ires bexprnd krange
```

```
kres bexprnd krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (*-krange* à *+krange*)

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode bexprnd. Il utilise le fichier *bexprnd.csd* [examples/bexprnd.csd].

### Exemple 57. Exemple de l'opcode bexprnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac          -iadc      ;;RT audio I/O
; -o bexprnd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a random number between -1 and 1.
; krange = 1
```

```
    i1 bexprnd 1

    print i1
    endin

; Instrument #2.
instr 2
; Generate a random number between -1 and 1.
; krange = 1

; *** Set seed from time system
seed 0

; *** Every run time different value
i1 bexprnd 1

    print i1
    endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 1.141
```

## Voir Aussi

*seed, betarand, cauchy, exprand, gauss, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.

# bformenc

bformenc — Obsolète. Encode un signal dans le format ambisonic B.

## Description

Encode un signal dans le format ambisonic B. Noter que cet opcode est obsolète et imprécis et qu'il est remplacé par l'opcode *bformenc1* bien meilleur qui reprend toutes les caractéristiques importantes ; noter que les arguments de gain ne sont pas disponibles dans *bformenc1*.

## Syntaxe

```
aw, ax, ay, az bformenc asig, kalpha, kbeta, kord0, kord1
```

```
aw, ax, ay, az, ar, as, at, au, av bformenc asig, kalpha, kbeta, \  
kord0, kord1, kord2
```

```
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc \  
asig, kalpha, kbeta, kord0, kord1, kord2, kord3
```

## Exécution

*aw, ax, ay, ...* -- cellules de sortie au format B.

*asig* -- signal d'entrée.

*kalpha* -- angle d'azimut en degrés (dans le sens des aiguilles d'une montre).

*kbeta* -- angle d'altitude en degrés.

*kord0* -- gain linéaire du format B d'ordre zéro.

*kord1* -- gain linéaire du format B du premier ordre.

*kord2* -- gain linéaire du format B du deuxième ordre.

*kord3* -- gain linéaire du format B du troisième ordre.

## Exemple

Voici un exemple de l'opcode bformenc. Il utilise le fichier *bformenc.csd* [examples/bformenc.csd].

### Exemple 58. Exemple de l'opcode bformenc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out   Audio in   No messages  
;-odac       -iadc       -d      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:
```

```

-o bformenc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 8

;bformenc is deprecated, please use bformenc1

instr 1
    ; generate pink noise
    anoise pinkish 1000

    ; two full turns
    kalpha line 0, p3, 720
    kbeta = 0

    ; fade ambisonic order from 2nd to 0th during second turn
    kord0 = 1
    kord1 linseg 1, p3 / 2, 1, p3 / 2, 0
    kord2 linseg 1, p3 / 2, 1, p3 / 2, 0

    ; generate B format
    aw, ax, ay, az, ar, as, at, au, av bformenc anoise, kalpha, kbeta, kord0, kord1, kord2

    ; decode B format for 8 channel circle loudspeaker setup
    a1, a2, a3, a4, a5, a6, a7, a8 bformdec 4, aw, ax, ay, az, ar, as, at, au, av

    ; write audio out
    outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 20 seconds.
i 1 0 20
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Samuel Groner  
2005

Nouveau dans la version 5.07. Obsolète dans la 5.09.



# bformenc1

bformenc1 — Encode un signal dans le format ambisonic B.

## Description

Encode un signal dans le format ambisonic B.

## Syntaxe

```
aw, ax, ay, az bformenc1 asig, kalpha, kbeta
```

```
aw, ax, ay, az, ar, as, at, au, av bformenc1 asig, kalpha, kbeta
```

```
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1 \  
    asig, kalpha, kbeta
```

## Exécution

*aw, ax, ay, ...* -- cellules de sortie au format B.

*asig* -- signal d'entrée.

*kalpha* -- angle d'azimut en degrés (dans le sens contraire des aiguilles d'une montre).

*kbeta* -- angle d'altitude en degrés.

## Exemple

Voici un exemple de l'opcode bformenc1. Il utilise le fichier *bformenc1.csd* [examples/bformenc1.csd].

### Exemple 59. Exemple de l'opcode bformenc1.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  No messages  
;-odac      -iadc      -d      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
-o bformenc.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
sr = 44100  
kr = 4410  
ksmps = 10  
nchnls = 8  
  
instr 1  
    ; generate pink noise  
    anoise pinkish 1000  
  
    ; two full turns  
    kalpha line 0, p3, 720
```

```
    kbeta = 0

    ; generate B format
    aw, ax, ay, az, ar, as, at, au, av bformenc1 anoise, kalpha, kbeta

    ; decode B format for 8 channel circle loudspeaker setup
    a1, a2, a3, a4, a5, a6, a7, a8 bformdec1 4, aw, ax, ay, az, ar, as, at, au, av

    ; write audio out
    outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

    ; Play Instrument #1 for 20 seconds.
    i 1 0 20
    e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*bformdec1*

## Crédits

Auteurs : Richard Furse, Bruce Wiggins et Fons Adriaensen, d'après le code de Samuel Groner 2008

Nouveau dans la version 5.09

# bformdec

bformdec — Obsolète. Décode un signal au format ambisonic B.

## Description

Décode un signal au format ambisonic B en signaux de haut-parleur spécifiques. Noter que cet opcode est obsolète et imprécis et qu'il est remplacé par l'opcode *bformdec1* bien meilleur qui reprend toutes les caractéristiques importantes.

## Syntaxe

```
ao1, ao2 bformdec isetup, aw, ax, ay, az [, ar, as, at, au, av \
    [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4 bformdec isetup, aw, ax, ay, az [, ar, as, at, \
    au, av [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4, ao5 bformdec isetup, aw, ax, ay, az [, ar, as, \
    at, au, av [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec isetup, aw, ax, ay, az \
    [, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]
```

## Initialisation

*isetup* — réglage de haut-parleur. Il y a cinq réglages possibles : 1 indique le réglage stéréo. Il doit y avoir deux cellules de sortie avec les positions de haut-parleur supposées valoir (330/0, 30/0).

2 indique le réglage quadraphonique. Il doit y avoir quatre cellules de sortie. Les positions de haut-parleur sont supposées valoir (45°/0), (135°/0), (225/0), (315/0).

3 est un réglage surround 5.1. Il doit y avoir cinq cellules de sortie. Le canal LFE n'est pas supporté. Les positions de haut-parleur sont supposées valoir (330/0), (30/0), (0/0), (250/0), (110/0).

4 indique huit haut-parleurs en cercle. Il doit y avoir huit cellules de sortie. Les positions de haut-parleur sont supposées valoir (22.5/0), (67.5/0), (112.5/0), (157.5/0), (202.5/0), (247.5/0), (292.5/0), (337.5/0).

5 indique un réglage cubique de huit haut-parleurs. Il doit y avoir huit cellules de sortie. Les positions de haut-parleur sont supposées valoir (45/0), (45/30), (135/0), (135/30), (225/0), (225/30), (315/0), (315/30).

## Exécution

*aw, ax, ay, ...* -- signal d'entrée au format B.

*ao1 .. ao8* — signaux de haut-parleur spécifiques en sortie.

## Exemple

Voici un exemple de l'opcode bformdec. Il utilise le fichier *bformenc.csd* [examples/bformenc.csd].

## Exemple 60. Exemple de l'opcode bformdec.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
;-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o bformenc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 8

;bformenc is deprecated, please use bformenc1

instr 1
; generate pink noise
anoise pinkish 1000

; two full turns
kalpha line 0, p3, 720
kbeta = 0

; fade ambisonic order from 2nd to 0th during second turn
kord0 = 1
kord1 linseg 1, p3 / 2, 1, p3 / 2, 0
kord2 linseg 1, p3 / 2, 1, p3 / 2, 0

; generate B format
aw, ax, ay, az, ar, as, at, au, av bformenc anoise, kalpha, kbeta, kord0, kord1, kord2

; decode B format for 8 channel circle loudspeaker setup
a1, a2, a3, a4, a5, a6, a7, a8 bformdec 4, aw, ax, ay, az, ar, as, at, au, av

; write audio out
outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 20 seconds.
i 1 0 20
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Samuel Groner  
2005

Nouveau dans la version 5.07. Obsolète dans la 5.09.

# bformdec1

bformdec1 — Décode un signal au format ambisonic B.

## Description

Décode un signal au format ambisonic B en signaux de haut-parleur spécifiques.

## Syntaxe

```
ao1, ao2 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, au, av \
[, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, \
au, av [, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4, ao5 bformdec1 isetup, aw, ax, ay, az [, ar, as, \
at, au, av [, abk, al, am, an, ao, ap, aq]]
```

```
ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec1 isetup, aw, ax, ay, az \
[, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]
```

## Initialisation

Noter que les angles horizontaux sont mesurés dans le sens contraire des aiguilles d'une montre dans cette description.

*isetup* — réglage de haut-parleur. Cinq réglages sont supportés :

- 1. Stéréo - L(90), R(-90) ; c'est un décodage stéréo dans le style M+S.
- 2. Quadraphonique - FL(45), BL(135), BR(-135), FR(-45). C'est un décodage "en phase" du premier ordre.
- 3. 5.0 - L(30), R(-30), C(0), BL(110), BR(-110). Noter que beaucoup de gens n'utilisent pas les angles ci-dessus pour leur grille de haut-parleurs et on peut réaliser un bon décodage pour DVD, etc en utilisant la configuration quadraphonique pour alimenter L, R, BL et BR (laissant C silencieux).
- 4. Octogone - FFL(22.5), FLL(67.5), BLL(112.5), BBL(157.5), BBR(-157.5), BRR(-112.5), FRR(-67.5), FFR(-22.5). C'est un décodage "en phase" de premier, deuxième ou troisième ordre en fonction du nombre de canaux en entrée.
- 5. Cube - FLD(45, -35.26), FLU(45, 35.26), BLD(135, -35.26), BLU(135, 35.26), BRD(-135, -35.26), BRU(-135, 35.26), FRD(-45, -35.26), FRU(-45, 35.26). C'est un décodage "en phase" du premier ordre.

## Exécution

*aw, ax, ay, ...* -- signal d'entrée au format B.

*ao1 .. ao8* — signaux de haut-parleur spécifiques en sortie.

## Exemple

Voici un exemple de l'opcode `bformdec1`. Il utilise le fichier `bformenc1.csd` [examples/bformenc1.csd].

### Exemple 61. Exemple de l'opcode `bformdec1`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
;-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o bformenc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 8

instr 1
; generate pink noise
anoise pinkish 1000

; two full turns
kalpha line 0, p3, 720
kbeta = 0

; generate B format
aw, ax, ay, az, ar, as, at, au, av bformenc1 anoise, kalpha, kbeta

; decode B format for 8 channel circle loudspeaker setup
a1, a2, a3, a4, a5, a6, a7, a8 bformdec1 4, aw, ax, ay, az, ar, as, at, au, av

; write audio out
outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 20 seconds.
i 1 0 20
e

</CsScore>
</CsSoundSynthesizer>
```

## Voir Aussi

*bformenc1*

## Credits

Auteurs : Richard Furse, Bruce Wiggins et Fons Adriaensen, d'après le code de Samuel Groner  
2008

Nouveau dans la version 5.09

# binit

binit — PVS tracks to amplitude+frequency conversion.

## Description

The binit opcode takes an input containing a TRACKS pv streaming signal (as generated, for instance by partials) and converts it into a equal-bandwidth bin-frame containing amplitude and frequency pairs (PVS\_AMP\_FREQ), suitable for overlap-add resynthesis (such as performed by pvsynth) or further PVS streaming phase vocoder signal transformations. For each frequency bin, it will look for a suitable track signal to fill it; if not found, the bin will be empty (0 amplitude). If more than one track fits a certain bin, the one with highest amplitude will be chosen. This means that not all of the input signal is actually 'binned', the operation is lossy. However, in many situations this loss is not perceptually relevant.

## Syntax

```
fsig binit fin, isize
```

## Performance

*fsig* -- output pv stream in PVS\_AMP\_FREQ format

*fin* -- input pv stream in TRACKS format

*isize* -- FFT size of output (N).

## Examples

### Exemple 62. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
fbins binit fst, 2048 ; convert it back to bins
aout pvsynth fbins ; overlap-add resynthesis
out aout
```

The example above shows partial tracking of an ifd-analysis signal, conversion to bin frames and overlap-add resynthesis.

## Credits

Author: Victor Lazzarini  
February 2006

New in Csound5.01

# biquad

biquad — A sweepable general purpose biquadratic digital filter.

## Description

A sweepable general purpose biquadratic digital filter.

## Syntax

```
ares biquad asig, kb0, kb1, kb2, ka0, ka1, ka2 [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- if non-zero, initialization will be skipped. Default value 0. (New in Csound version 3.50)

## Performance

*asig* -- input signal

*biquad* is a general purpose biquadratic digital filter of the form:

$$a0*y(n) + a1*y[n-1] + a2*y[n-2] = b0*x[n] + b1*x[n-1] + b2*x[n-2]$$

This filter has the following frequency response:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + b2*Z^{-2}}{a0 + a1*Z^{-1} + a2*Z^{-2}}$$

This type of filter is often encountered in digital signal processing literature. It allows six user-defined k-rate coefficients.

## Examples

Here is an example of the biquad opcode. It uses the file *biquad.csd* [examples/biquad.csd].

### Exemple 63. Example of the biquad opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>  
<CsOptions>
```



```

; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o biquad.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1.
instr 1
; Get the values from the score.
idur = p3
iamp = p4
icps = cpspch(p5)
kfco = p6
krez = p7

; Calculate the biquadratic filter's coefficients
kfcon = 2*3.14159265*kfco/sr
kalpha = 1-2*krez*cos(kfcon)*cos(kfcon)+krez*krez*cos(2*kfcon)
kbeta = krez*krez*sin(2*kfcon)-2*krez*cos(kfcon)*sin(kfcon)
kgama = 1+cos(kfcon)
km1 = kalpha*kgama+kbeta*sin(kfcon)
km2 = kalpha*kgama-kbeta*sin(kfcon)
kden = sqrt(km1*km1+km2*km2)
kb0 = 1.5*(kalpha*kalpha+kbeta*kbeta)/kden
kb1 = kb0
kb2 = 0
ka0 = 1
ka1 = -2*krez*cos(kfcon)
ka2 = krez*krez

; Generate an input signal.
axn vco 1, icps, 1

; Filter the input signal.
ayn biquad axn, kb0, kb1, kb2, ka0, ka1, ka2
outs ayn*iamp/2, ayn*iamp/2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

;      Sta  Dur  Amp  Pitch Fco  Rez
i 1  0.0  1.0  20000  6.00 1000  .8
i 1  1.0  1.0  20000  6.03 2000  .95
e

</CsScore>
</CsSoundSynthesizer>

```

Here is another example of the biquad opcode used for modal synthesis. It uses the file *biquad-2.csd* [examples/biquad-2.csd]. See the *Modal Frequency Ratios* appendix for other frequency ratios.

## Exemple 64. Example of the biquad opcode for modal synthesis.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o biquad-2.wav -W ;; for file output any platform

```

```

</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

/* modal synthesis using biquad filters as oscillators
   Example by Scott Lindroth 2007 */

instr 1

    ipi = 3.1415926
    idenom = sr*0.5

    ipulseSpd = p4
    icps      = p5
    ipan      = p6
    iamp      = p7
    iModes    = p8

    apulse    mpulse iamp, 0

    icps      = cpspch( icps )

    ; filter gain

    iamp1 = 600
    iamp2 = 1000
    iamp3 = 1000
    iamp4 = 1000
    iamp5 = 1000
    iamp6 = 1000

    ; resonance

    irpole1 = 0.99999
    irpole2 = irpole1
    irpole3 = irpole1
    irpole4 = irpole1
    irpole5 = irpole1
    irpole6 = irpole1

    ; modal frequencies

    if (iModes == 1) goto modes1
    if (iModes == 2) goto modes2

modes1:
    if1 = icps * 1           ;pot lid
    if2 = icps * 6.27
    if3 = icps * 3.2
    if4 = icps * 9.92
    if5 = icps * 14.15
    if6 = icps * 6.23
    goto nextPart

modes2:
    if1 = icps * 1           ;uniform wood bar
    if2 = icps * 2.572
    if3 = icps * 4.644
    if4 = icps * 6.984
    if5 = icps * 9.723
    if6 = icps * 12.0
    goto nextPart

nextPart:

    ; convert frequency to radian frequency

    itheta1 = (if1/idenom) * ipi
    itheta2 = (if2/idenom) * ipi
    itheta3 = (if3/idenom) * ipi
    itheta4 = (if4/idenom) * ipi
    itheta5 = (if5/idenom) * ipi
    itheta6 = (if6/idenom) * ipi

    ; calculate coefficients

    ibl1 = -2 * irpole1 * cos(itheta1)

```

```

ib21 = irpole1 * irpole1
ib12 = -2 * irpole2 * cos(itheta2)
ib22 = irpole2 * irpole2
ib13 = -2 * irpole3 * cos(itheta3)
ib23 = irpole3 * irpole3
ib14 = -2 * irpole4 * cos(itheta4)
ib24 = irpole4 * irpole4
ib15 = -2 * irpole5 * cos(itheta5)
ib25 = irpole5 * irpole5
ib16 = -2 * irpole6 * cos(itheta6)
ib26 = irpole6 * irpole6

;printk 1, ib 11
;printk 1, ib 21

; also try setting the -1 coeff. to 0, but be sure to scale down the amplitude!

asin1      biquad  apulse * iamp1, 1, 0, -1, 1, ib11, ib21
asin2      biquad  apulse * iamp2, 1, 0, -1, 1, ib12, ib22
asin3      biquad  apulse * iamp3, 1, 0, -1, 1, ib13, ib23
asin4      biquad  apulse * iamp4, 1, 0, -1, 1, ib14, ib24
asin5      biquad  apulse * iamp5, 1, 0, -1, 1, ib15, ib25
asin6      biquad  apulse * iamp6, 1, 0, -1, 1, ib16, ib26

afin      =      (asin1 + asin2 + asin3 + asin4 + asin5 + asin6)

outs      afin * sqrt(p6), afin*sqrt(1-p6)

endin
</CsInstruments>
<CsScore>
;ins      st      dur      pulseSpd      pch      pan      amp      Modes
i1        0      12      0      7.089      0      0.7      2
i1        .      .      .      7.09      1      .      .
i1        .      .      .      7.091      0.5      .      .

i1        0      12      0      8.039      0      0.7      2
i1        0      12      0      8.04      1      0.7      2
i1        0      12      0      8.041      0.5      0.7      2

i1        9      .      .      7.089      0      .      2
i1        .      .      .      7.09      1      .      .
i1        .      .      .      7.091      0.5      .      .

i1        9      12      0      8.019      0      0.7      2
i1        9      12      0      8.02      1      0.7      2
i1        9      12      0      8.021      0.5      0.7      2
e
</CsScore>
</CsoundSynthesizer>

```

## See Also

*biquada, moogvcf, rezy*

## Credits

Author: Hans Mikelson  
October 1998

New in Csound version 3.49

# biquada

biquada — A sweepable general purpose biquadratic digital filter with a-rate parameters.

## Description

A sweepable general purpose biquadratic digital filter.

## Syntax

```
ares biquada asig, ab0, ab1, ab2, aa0, aa1, aa2 [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- if non-zero, initialization will be skipped. Default value 0. (New in Csound version 3.50)

## Performance

*asig* -- input signal

*biquada* is a general purpose biquadratic digital filter of the form:

$$a0*y(n) + a1*y[n-1] + a2*y[n-2] = b0*x[n] + b1*x[n-1] + b2*x[n-2]$$

This filter has the following frequency response:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + b2*Z^{-2}}{a0 + a1*Z^{-1} + a2*Z^{-2}}$$

This type of filter is often encountered in digital signal processing literature. It allows six user-defined a-rate coefficients.

## See Also

*biquad*

## Credits

Author: Hans Mikelson  
October 1998

New in Csound version 3.49

# birnd

birnd — Retourne un nombre aléatoire dans un intervalle bipolaire.

## Description

Retourne un nombre aléatoire dans un intervalle bipolaire.

## Syntaxe

**birnd**(x) (taux-i ou -k seulement)

Où l'argument entre parenthèses peut être une expression. Ces convertisseurs de valeur échantillonnent une séquence aléatoire globale, mais sans référencer une *racine*. Le résultat peut devenir un terme d'une expression ultérieure.

## Exécution

Retourne un nombre aléatoire dans l'intervalle bipolaire allant de  $-x$  à  $x$ . *rnd* et *birnd* obtiennent leurs valeurs d'un générateur de nombres pseudo-aléatoires global, puis les mettent à l'échelle de l'intervalle demandé. Le générateur global unique distribuera ainsi sa séquence à ces unités durant toute l'exécution, quelque soit l'ordre d'arrivée de ces demandes.

## Exemples

Voici un exemple de l'opcode birnd. Il utilise le fichier *birnd.csd* [examples/birnd.csd].

### Exemple 65. Exemple de l'opcode birnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o birnd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a random number from -1 to 1.
i1 = birnd(1)
print i1
endin

</CsInstruments>
```

```
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #1 for one second.
i 1 1 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  i1 = 0.947
instr 1:  i1 = -0.721
```

## Voir Aussi

*rnd*

## Crédits

Auteur: Barry L. Vercoe  
MIT  
Cambridge, Massachussetts  
1997

Etendu dans la version 3.47 au taux-x par John ffitch.

Exemple écrit par Kevin Conder.

# bqrez

bqrez — A second-order multi-mode filter.

## Description

A second-order multi-mode filter.

## Syntax

```
ares bqrez asig, xfco, xres [, imode] [, iskip]
```

## Initialization

*imode* (optional, default=0) -- The mode of the filter. Choose from one of the following:

- 0 = low-pass (default)
- 1 = high-pass
- 2 = band-pass
- 3 = band-reject
- 4 = all-pass

*iskip* (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

## Performance

*ares* -- output audio signal.

*asig* -- input audio signal.

*xfco* -- filter cut-off frequency in Hz. May be i-time, k-rate, a-rate.

*xres* -- amount of resonance. Values of 1 to 100 are typical. Resonance should be one or greater. A value of 100 gives a 20dB gain at the cutoff frequency. May be i-time, k-rate, a-rate.

All filter modes can be frequency modulated as well as the resonance can also be frequency modulated.

*bqrez* is a resonant low-pass filter created using the Laplace s-domain equations for low-pass, high-pass, and band-pass filters normalized to a frequency. The bi-linear transform was used which contains a frequency transform constant from s-domain to z-domain to exactly match the frequencies together. A lot of trigonometric identities were used to simplify the calculation. It is very stable across the working frequency range up to the Nyquist frequency.

## Examples

Here is an example of the *bqrez* opcode. It uses the file *bqrez.csd* [examples/bqrez.csd].

## Exemple 66. Example of the bqrez opcode borrowed from the « rezzy » opcode in Kevin Conder's manual.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o bqrez.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Gerassimof from example by Kevin Conder */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a nice sawtooth waveform.
asig vco 16000, 220, 1

; Vary the filter-cutoff frequency from .2 to 2 KHz.
kfco line 200, p3, 2000

; Set the resonance amount.
kres init 0.99

a1 bqrez asig, kfco, kres

out a1
endin

</CsInstruments>
<CsScore>

/* Written by Matt Gerassimof from example by Kevin Conder */
; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*biquad, moogvcf, rezzy*

## Credits

Author: Matt Gerassimoff  
 New in version 4.32  
 Written in November 2002.



# butbp

butbp — Same as the butterbp opcode.

## Description

Same as the *butterbp* opcode.

## Syntax

```
ares butbp asig, kfreq, kband [, iskip]
```

# butbr

butbr — Same as the butterbr opcode.

## Description

Same as the *butterbr* opcode.

## Syntax

```
ares butbr asig, kfreq, kband [, iskip]
```

# buthp

buthp — Same as the butterhp opcode.

## Description

Same as the *butterhp* opcode.

## Syntax

```
ares buthp asig, kfreq [, iskip]
```

# butlp

butlp — Same as the butterlp opcode.

## Description

Same as the *butterlp* opcode.

## Syntax

```
ares butlp asig, kfreq [, iskip]
```

# butterbp

butterbp — A band-pass Butterworth filter.

## Description

Implementation of a second-order band-pass Butterworth filter. This opcode can also be written as *butbp*.

## Syntax

```
ares butterbp asig, kfreq, kband [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- Skip initialization if present and non-zero.

## Performance

These filters are Butterworth second-order IIR filters. They are slightly slower than the original filters in Csound, but they offer an almost flat passband and very good precision and stopband attenuation.

*asig* -- Input signal to be filtered.

*kfreq* -- Cutoff or center frequency for each of the filters.

*kband* -- Bandwidth of the bandpass and bandreject filters.

## Examples

Here is an example of the butterbp opcode. It uses the file *butterbp.csd* [examples/butterbp.csd].

### Exemple 67. Example of the butterbp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o butterbp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
```

```
; White noise signal
asig rand 22050

out asig
endin

; Instrument #2 - a filtered noise waveform.
instr 2
; White noise signal
asig rand 22050

; Filter it, passing only 1950 to 2050 Hz.
abp butterbp asig, 2000, 100

out abp
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*butterbr, butterhp, butterlp*

## Credits

Author: Paris Smaragdis  
MIT, Cambridge  
1995

Existed in 3.30

# butterbr

butterbr — A band-reject Butterworth filter.

## Description

Implementation of a second-order band-reject Butterworth filter. This opcode can also be written as *but-br*.

## Syntax

```
ares butterbr asig, kfreq, kband [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- Skip initialization if present and non-zero.

## Performance

These filters are Butterworth second-order IIR filters. They are slightly slower than the original filters in Csound, but they offer an almost flat passband and very good precision and stopband attenuation.

*asig* -- Input signal to be filtered.

*kfreq* -- Cutoff or center frequency for each of the filters.

*kband* -- Bandwidth of the bandpass and bandreject filters.

## Examples

Here is an example of the butterbr opcode. It uses the file *butterbr.csd* [examples/butterbr.csd].

### Exemple 68. Example of the butterbr opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o butterbr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
```

```
; White noise signal
asig rand 22050

out asig
endin

; Instrument #2 - a filtered noise waveform.
instr 2
; White noise signal
asig rand 22050

; Filter it, cutting 2000 to 6000 Hz.
abr butterbr asig, 4000, 2000

out abr
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*butterbp, butterhp, butterlp*

## Credits

Author: Paris Smaragdis  
MIT, Cambridge  
1995

Existed in 3.30



# butterhp

butterhp — A high-pass Butterworth filter.

## Description

Implementation of second-order high-pass Butterworth filter. This opcode can also be written as *buthp*.

## Syntax

```
ares butterhp asig, kfreq [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- Skip initialization if present and non-zero.

## Performance

These filters are Butterworth second-order IIR filters. They are slightly slower than the original filters in Csound, but they offer an almost flat passband and very good precision and stopband attenuation.

*asig* -- Input signal to be filtered.

*kfreq* -- Cutoff or center frequency for each of the filters.

## Examples

Here is an example of the butterhp opcode. It uses the file *butterhp.csd* [examples/butterhp.csd].

### Exemple 69. Example of the butterhp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent
-odac      -iadc      -d      ;;realtime output
; For Non-realtime ouput leave only the line below:
; -o butterhp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
; White noise signal
asig rand 22050

out asig
```

```
endin

; Instrument #2 - a filtered noise waveform.
instr 2
; White noise signal
asig rand 22050

; Filter it, passing frequencies above 250 Hz.
ahp butterhp asig, 250

out ahp
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*butterbp, butterbr, butterlp*

## Credits

Author: Paris Smaragdis  
MIT, Cambridge  
1995

Existed in 3.30

# butterlp

butterlp — A low-pass Butterworth filter.

## Description

Implementation of a second-order low-pass Butterworth filter. This opcode can also be written as *butlp*.

## Syntax

```
ares butterlp asig, kfreq [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- Skip initialization if present and non-zero.

## Performance

These filters are Butterworth second-order IIR filters. They are slightly slower than the original filters in Csound, but they offer an almost flat passband and very good precision and stopband attenuation.

*asig* -- Input signal to be filtered.

*kfreq* -- Cutoff or center frequency for each of the filters.

## Examples

Here is an example of the butterlp opcode. It uses the file *butterlp.csd* [examples/butterlp.csd].

### Exemple 70. Example of the butterlp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o butterlp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
; White noise signal
asig rand 22050

out asig
```

```
    endin

    ; Instrument #2 - a filtered noise waveform.
    instr 2
    ; White noise signal
    asig rand 22050

    ; Filter it, cutting frequencies above 1 KHz.
    alp butterlp asig, 1000

    out alp
    endin

</CsInstruments>
<CsScore>

    ; Play Instrument #1 for two seconds.
    i 1 0 2
    ; Play Instrument #2 for two seconds.
    i 2 2 2
    e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*butterbp, butterbr, butterhp*

## Credits

Author: Paris Smaragdis  
MIT, Cambridge  
1995

Existed in 3.30

# button

button — Contrôles sur l'écran.

## Description

Contrôles sur l'écran. Nécessite Winsound ou TCL/TK.

## Syntaxe

`kres button knum`

## Exécution

*kres* -- valeur du contrôle bouton. Si le bouton a été enfoncé depuis la dernière k-période, retourne 1, sinon 0.

*knun* -- le numéro du bouton. S'il n'existe pas, il apparaît sur l'écran à l'initialisation.

## Voir Aussi

*checkbox*

## Credits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Septembre 2000

Nouveau dans la version 4.08 de Csound

# buzz

**buzz** — La sortie est un ensemble de partiels sinus en relation harmonique.

## Description

La sortie est un ensemble de partiels sinus en relation harmonique.

## Syntaxe

```
ares buzz xamp, xcps, knh, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table d'une fonction stockée contenant une onde sinus. Une grande table d'au moins 8192 points est recommandée.

*iphs* (facultatif, par défaut 0) -- phase initiale de la fréquence fondamentale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

## Exécution

*xamp* -- amplitude

*xcps* -- fréquence en cycles par seconde

Les unités *buzz* génèrent un ensemble additif de partiels cosinus en relation harmonique de fréquence fondamentale *xcps*, et dont les amplitudes sont pondérées de telle façon que la crête de leur somme égale *xamp*. Le choix et l'importance des partiels sont déterminés par les paramètres de contrôle suivants :

*knh* -- nombre total d'harmoniques demandés. Nouveau dans la version 3.57 de Csound, *knh* vaut un par défaut. Si *knh* est négatif, sa valeur absolue est utilisée.

*buzz* et *gbuzz* sont utiles comme sources de son complexe dans la synthèse soustractive. *buzz* est un cas particulier du plus général *gbuzz* dans lequel  $klh = kmul = 1$  ; il produit ainsi un ensemble de *knh* harmoniques de même importance, commençant avec le fondamental. (C'est un train d'impulsions à bande de fréquence limitée ; si les partiels vont jusqu'à la fréquence de Nyquist, c'est-à-dire  $knh = \text{int}(sr / 2 / \text{fréq. fondamentale})$ , le résultat est un train d'impulsions réelles d'amplitude *xamp*.)

Bien que l'on puisse faire varier *knh* durant l'exécution, sa valeur interne est nécessairement un entier ce qui peut provoquer des « pops » dûs à des discontinuités dans la sortie. *buzz* peut être modulé en amplitude et/ou en fréquence soit par des signaux de contrôle soit par des signaux audio.

Nota Bene : cette unité a son pendant avec *GENII*, dans lequel le même ensemble de cosinus peut être stocké dans une table de fonction qui sera lue par un oscillateur. Bien que plus efficace en termes de calcul, le train d'impulsions stocké a un contenu spectral fixe, non variable dans le temps comme celui décrit ci-dessus.

## Exemples

Voici un exemple de l'opcode *buzz*. Il utilise le fichier *buzz.csd* [examples/buzz.csd].

## Exemple 71. Exemple de l'opcode buzz.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o buzz.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 20000
  kcps = 440
  knh = 3
  ifn = 1

  a1 buzz kamp, kcps, knh, ifn
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*gbuzz*

## Crédits

Septembre 2003. Merci à Kanata Motohashi pour avoir corrigé les mentions du paramètre *kmul*.

Exemple écrit par Kevin Conder.

# cabasa

cabasa — Modèle semi-physique d'un son de cabasa.

## Description

*cabasa* est un modèle semi-physique d'un son de cabasa. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares cabasa iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont de type stochastique, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 512.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0.998 + (\text{idamp} * 0.002)$$

La valeur par défaut de *damping\_amount* est 0,997 ce qui signifie que la valeur par défaut de *idamp* est - 0,5. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode cabasa. Il utilise le fichier *cabasa.csd* [examples/cabasa.csd].

### Exemple 72. Exemple de l'opcode cabasa.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;;RT audio I/O
```



```

; For Non-realtime ouput leave only the line below:
; -o cabasa.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra -----

    sr =          44100
    kr =          4410
    ksmps =        10
    nchnls =        1

    instr 01          ;an example of a cabasa
a1      cabasa p4, 0.01
        out a1
        endin

</CsInstruments>
<CsScore>

;score -----

i1 0 1 26000
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*crunch, sandpaper, sekere, stix*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)  
 Adapté par John ffitich  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# cauchy

cauchy — Générateur de nombres aléatoires de distribution de Cauchy.

## Description

Générateur de nombres aléatoires de distribution de Cauchy. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares cauchy kalpha
```

```
ires cauchy kalpha
```

```
kres cauchy kalpha
```

## Exécution

*kalpha* -- contrôle la dispersion centrée sur zéro (un grand *kalpha* = une grande dispersion). Donne en sortie des nombres positifs et négatifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode cauchy. Il utilise le fichier *cauchy.csd* [examples/cauchy.csd].

### Exemple 73. Exemple de l'opcode cauchy.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac          -iadc      ;;RT audio I/O
; -o cauchy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a random number, spread from 10.
; kalpha = 10
```

```

i1 cauchy 10

print i1
endin

; Instrument #2.
instr 2
; Generate a random number, spread from 10.
; kalpha = 10

seed 0

i1 cauchy 10

print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1: i1 = -0.106
```

## Voir Aussi

*seed, betarand, bexprnd, exprand, gauss, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Existait dans la 3.30

Exemple écrit par Kevin Conder.

# ceil

ceil — Retourne le plus petit entier supérieur ou égal à  $x$ .

## Description

Retourne le plus petit entier supérieur ou égal à  $x$ .

## Syntaxe

`ceil(x)` (argument au taux d'initialisation, de contrôle ou audio)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Voir Aussi

*abs, exp, int, log, log10, i, sqrt*

## Crédits

Auteur : Istvan Varga  
Nouveau dans Csound 5  
2005

# cent

cent — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de cents.

## Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de cents.

## Syntaxe

`cent ( x )`

Cette fonction travaille aux taux-i, -k et -a.

## Initialisation

*x* -- une valeur exprimée en cents.

## Exécution

La valeur retournée par la fonction *cent* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre de cents spécifié.

## Exemples

Voici un exemple de l'opcode *cent*. Il utilise le fichier *cent.csd* [examples/cent.csd].

### Exemple 74. Exemple de l'opcode cent.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cent.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; The root note is A above middle-C (440 Hz)
iroot = 440

; Raise the root note by 300 cents to C.
icents = 300

; Calculate the new note.
```

```
ifactor = cent(icents)
inew = iroot * ifactor

; Print out of all of the values.
print iroot
print ifactor
print inew
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1: iroot = 440.000
instr 1: ifactor = 1.189
instr 1: inew = 523.229
```

## Voir Aussi

*db, octave, semitone*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.16

# cggoto

cggoto — Transfert conditionnel du contrôle à chaque passage.

## Description

Transfère le contrôle vers *label* à chaque passage. (Combinaison de *cigoto* et de *ckgoto*)

## Syntaxe

`cggoto condition, label`

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression, et où *condition* utilise un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exemples

Voici un exemple de l'opcode cggoto. Il utilise le fichier *cggoto.csd* [examples/cggoto.csd].

### Exemple 75. Exemple de l'opcode cggoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out   Audio in   No messages
-odac         -iadc      -d          ;;RT audio I/O
; -o cggoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  il = p4

  ; If il is equal to one, play a high note.
  ; Otherwise play a low note.
  cggoto (il == 1), highnote

lownote:
  a1 oscil 10000, 220, 1
  goto playit

highnote:
  a1 oscil 10000, 440, 1
  goto playit

playit:
  out a1
endin

</CsInstruments>
<CsScore>
```

```
; Table #1: a simple sine wave.  
f 1 0 32768 10 1  
  
; Play lownote for one second.  
i 1 0 1 1  
; Play highnote for one second.  
i 1 1 1 2  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*cigoto, ckgoto, cngoto, if, igoto, kgoto, tigoto, timout*

## Crédits

Exemple écrit par Kevin Conder.



# chanctrl

chanctrl — Prend la valeur actuelle d'un contrôleur d'un canal MIDI.

## Description

Prend la valeur actuelle d'un contrôleur et le configure optionnellement dans un intervalle spécifié.

## Syntaxe

```
ival chanctrl ichnl, ictlno [, ilow] [, ihigh]
```

```
kval chanctrl ichnl, ictlno [, ilow] [, ihigh]
```

## Initialisation

*ichnl* -- le canal MIDI (1-16).

*ictlno* -- le numéro du contrôleur MIDI (0-127).

*ilow, ihigh* -- Limites inférieure et supérieure de la configuration

## Crédits

Auteur : Mike Berry  
Collège Mills  
Mai, 1997

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# changed

changed — k-rate signal change detector.

## Description

This opcode outputs a trigger signal that informs when any one of its k-rate arguments has changed. Useful with valuator widgets or MIDI controllers.

## Syntax

```
ktrig changed kvar1 [, kvar2,..., kvarN]
```

## Performance

*ktrig* - Outputs a value of 1 when any of the k-rate signals has changed, otherwise outputs 0.

*kvar1* [, *kvar2*,..., *kvarN*] - k-rate variables to watch for changes.

## Examples

Here is an example of the changed opcode. It uses the file *changed.csd* [examples/changed.csd].

### Exemple 76. Example of the changed opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      100
ksmps   =      441
nchnls  =      2

      instr      1
ksig oscil 2,0.5,1
kint = int(ksig)
ktrig changed kint
printk 0.2, kint
printk2 ktrig
      endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 20

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado.

Example written by Andrés Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# chani

chani — Lit des données depuis le bus logiciel.

## Description

Lit des données depuis un canal du bus logiciel entrant.

## Syntaxe

```
kval chani kchan
```

```
aval chani kchan
```

## Exécution

*kchan* -- un entier positif indiquant quel canal du bus logiciel lire.

Noter que les bus logiciels entrant et sortant sont indépendants et qu'ils ne sont pas des bus mélangeurs. De plus, les bus au taux-k et au taux-a sont indépendants. La dernière valeur reste disponible jusqu'à ce qu'une nouvelle valeur soit écrite. Il n'y a pas de limite imposée au nombre de bus mais comme ils consomment de la mémoire, il est préférable d'en utiliser un petit nombre.

## Exemple

L'exemple montre l'utilisation du bus logiciel comme signal de contrôle asynchrone pour fixer la fréquence de coupure du filtre. On suppose qu'un programme externe utilisant l'API fournit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

instr 1
  kc chani 1
  a1 oscil p4, p5, 100
  a2 lowpass2 a1, kc, 200
  out
endin
```

## Crédits

Auteur : John fitch  
2005

Nouveau dans Csound 5.00

# chano

chano — Envoie des données vers le bus logiciel sortant.

## Description

Envoie des données vers un canal du bus logiciel sortant.

## Syntaxe

```
chano kval, kchan
```

```
chano aval, kchan
```

## Exécution

*xval* --- valeur à transmettre.

*kchan* -- un entier positif indiquant sur quel canal du bus logiciel écrire.

Noter que les bus logiciels entrant et sortant sont indépendants et qu'ils ne sont pas des bus mélangeurs. De plus, les bus au taux-k et au taux-a sont indépendants. La dernière valeur reste disponible jusqu'à ce qu'une nouvelle valeur soit écrite. Il n'y a pas de limite imposée au nombre de bus mais comme ils consomment de la mémoire, il est préférable d'en utiliser un petit nombre.

## Exemple

L'exemple montre l'utilisation du bus logiciel comme canal de sortie d'un signal audio. On suppose qu'un programme externe utilisant l'API reçoit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

instr 1
  al oscil p4, p5, 100
    chano 1, al
endin
```

## Crédits

Auteur : John fitch  
2005

Nouveau dans Csound 5.00

# chebyshevpoly

chebyshevpoly — Efficiently evaluates the sum of Chebyshev polynomials of arbitrary order.

## Description

The *chebyshevpoly* opcode calculates the value of a polynomial expression with a single a-rate input variable that is made up of a linear combination of the first N Chebyshev polynomials of the first kind. Each Chebyshev polynomial,  $T_n(x)$ , is weighted by a k-rate coefficient,  $kn$ , so that the opcode is calculating a sum of any number of terms in the form  $kn * T_n(x)$ . Thus, the *chebyshevpoly* opcode allows for the waveshaping of an audio signal with a *dynamic* transfer function that gives precise control over the harmonic content of the output.

## Syntax

```
aout chebyshevpoly ain, k0 [, k1 [, k2 [...]]]
```

## Performance

*ain* -- the input signal used as the independent variable of the Chebyshev polynomials ("x").

*aout* -- the output signal ("y").

*k0, k1, k2, ...* -- k-rate multipliers for each Chebyshev polynomial.

This opcode is very useful for dynamic waveshaping of an audio signal. Traditional waveshaping techniques utilize a lookup table for the transfer function -- usually a sum of Chebyshev polynomials. When a sine wave at full-scale amplitude is used as an index to read the table, the precise harmonic spectrum as defined by the weights of the Chebyshev polynomials is produced. A dynamic spectrum is achieved by varying the amplitude of the input sine wave, but this produces a non-linear change in the spectrum.

By directly calculating the Chebyshev polynomials, the *chebyshevpoly* opcode allows more control over the spectrum and the number of harmonic partials added to the input can be varied with time. The value of each  $kn$  coefficient directly controls the amplitude of the  $n$ th harmonic partial if the input *ain* is a sine wave with amplitude = 1.0. This makes *chebyshevpoly* an efficient additive synthesis engine for N partials that requires only one oscillator instead of N oscillators. The amplitude or waveform of the input signal can also be changed for different waveshaping effects.

If we consider the input parameter *ain* to be "x" and the output *aout* to be "y", then the *chebyshevpoly* opcode calculates the following equation:

$$y = k0 * T0(x) + k1 * T1(x) + k2 * T2(x) + k3 * T3(x) + \dots$$

where the  $T_n(x)$  are defined by the recurrence relation

$$\begin{aligned} T0(x) &= 1, \\ T1(x) &= x, \\ Tn(x) &= 2x * T[n-1](x) - T[n-2](x) \end{aligned}$$

More information about Chebyshev polynomials can be found on Wikipedia at [http://en.wikipedia.org/wiki/Chebyshev\\_polynomial](http://en.wikipedia.org/wiki/Chebyshev_polynomial) [http://en.wikipedia.org/wiki/Chebyshev\_polynomial]

## See Also

*polynomial, mac maca*

## Examples

Here is an example of the `chebyshevpoly` opcode. It uses the file `chebyshevpoly.csd` [examples/chebyshevpoly.csd].

### Exemple 77. Example of the chebyshevpoly opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441

; time-varying mixture of first six harmonics
instr 1
; According to the GEN13 manual entry,
; the pattern + - - + + - - for the signs of
; the chebyshev coefficients has nice properties.

; these six lines control the relative powers of the harmonics
k1      line      1.0, p3, 0.0
k2      line      -0.5, p3, 0.0
k3      line      -0.333, p3, -1.0
k4      line      0.0, p3, 0.5
k5      line      0.0, p3, 0.7
k6      line      0.0, p3, -1.0

; play the sine wave at a frequency of 256 Hz
ax      oscili    1.0, 256, 1

; waveshape it
ay      chebyshevpoly ax, 0, k1, k2, k3, k4, k5, k6

; avoid clicks, scale final amplitude, and output
adeclick linseg    0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0
ay      out       ay * adeclick * 10000
endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1 ; a sine wave

i1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Anthony Kozar  
January 2008

New in Csound version 5.08



# checkbox

checkbox — Sense on-screen controls.

## Description

Sense on-screen controls. Requires Winsound or TCL/TK.

## Syntax

```
kres checkbox knum
```

## Performance

*kres* -- value of the checkbox control. If the checkbox is set (pushed) then return 1, if not, return 0.

*knun* -- the number of the checkbox. If it does not exist, it is made on-screen at initialization.

## Examples

Here is a simple example of the checkbox opcode. It uses the file *checkbox.csd* [examples/checkbox.csd].

### Exemple 78. Simple example of the checkbox opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o checkbox.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1
; Get the value from the checkbox.
k1 checkbox 1

; If the checkbox is selected then k2=440, otherwise k2=880.
k2 = (k1 == 0 ? 440 : 880)

a1 oscil 10000, k2, 1
out a1
endin

</CsInstruments>
<CsScore>
```

```
; Just generate a nice, ordinary sine wave.  
f 1 0 32768 10 1  
  
; Play Instrument #1 for ten seconds.  
i 1 0 10  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## See Also

*button*

## Credits

Author: John ffitch  
University of Bath, Codemist. Ltd.  
Bath, UK  
September, 2000

Example written by Kevin Conder.

New in Csound version 4.08

# chn

chn — Déclare un canal du bus logiciel nommé.

## Description

Déclare un canal du bus logiciel nommé, en donnant des paramètres facultatifs dans le cas d'un canal de contrôle. Si le canal n'existe pas encore, il est créé avec une valeur initiale de zéro ou une chaîne de caractères vide. Sinon le type (contrôle, audio ou chaînes de caractères) du canal existant doit correspondre à la déclaration ou bien il y aura une erreur d'initialisation. Le mode entrée/sortie d'un canal existant est mis à jour de façon à représenter le OU binaire entre la valeur précédente et la nouvelle.

## Syntaxe

```
chn_k Sname, imode[, itype, idflt, imin, imax]
```

```
chn_a Sname, imode
```

```
chn_s Sname, imode
```

## Initialisation

*imode* -- somme d'au moins un des nombres suivants : 1 pour entrée et 2 pour sortie.

*itype* (facultatif, 0 par défaut) -- sous-type du canal, seulement pour les canaux de contrôle. Les valeurs possibles sont :

- 0 : par défaut / indéfini (*idflt*, *imin* et *imax* sont ignorés)
- 1 : seulement des valeurs entières
- 2 : échelle linéaire
- 3 : échelle exponentielle

*idflt* (facultatif, 0 par défaut) -- valeur par défaut, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être supérieur ou égal à *imin* et inférieur ou égal à *imax*.

*imin* (facultatif, 0 par défaut) -- valeur minimale, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être différent de zéro pour l'échelle exponentielle (*itype* = 3).

*imax* (facultatif, 0 par défaut) -- valeur maximale, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être supérieur à *imin*. Dans le cas d'une échelle exponentielle, il doit également avoir le même signe que *imin*.

## Notes

Les paramètres du canal (*imode*, *itype*, *idflt*, *imin* et *imax*) ne sont que des indications pour l'application hôte ou un logiciel externe accédant au bus par l'API, et ils ne restreignent en rien la lecture ou l'écriture sur le canal. De plus, la valeur initiale d'un nouveau canal de contrôle est zéro, quelque soit la valeur de *idflt*.

Il peut être préférable d'utiliser *chnexport* pour communiquer avec un logiciel externe, car il permet un accès direct aux variables de l'orchestre exportées comme des canaux du bus, ce qui évite l'utilisation de *chnset* et de *chnget* pour envoyer ou recevoir des données.

## Exécution

**chn\_k**, **chn\_a**, et **chn\_S** déclarent respectivement un canal de contrôle, un canal audio ou un canal de chaînes de caractères.

## Exemple

L'exemple montre l'utilisation du bus logiciel comme signal de contrôle asynchrone pour fixer la fréquence de coupure du filtre. On suppose qu'un programme externe utilisant l'API fournit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

chn_k "cutoff", 1, 3, 1000, 500, 2000

instr 1
  kc chnget "cutoff"
  a1 oscil p4, p5, 100
  a2 lowpass2 a1, kc, 200
  out
endin
```

## Crédits

Auteur : Istvan Varga  
2005

# chnclear

chnclear — Efface un canal de sortie audio du bus logiciel nommé.

## Description

Met à zéro un canal de sortie audio du bus logiciel nommé. Cela implique une déclaration du canal avec *imode=2* (voir aussi *chn\_a*).

## Syntaxe

`chnclear Sname`

## Initialisation

*Sname* -- une chaîne de caractères indiquant quel canal du bus logiciel effacer.

## Crédits

Auteur : Istvan Varga  
2006

# chnexport

chnexport — Exporte une variable globale en tant que canal du bus.

## Description

Exporte une variable globale en tant que canal du bus ; le canal ne doit pas déjà exister sinon il y aura une erreur d'initialisation. On appelle normalement cet opcode depuis l'en-tête de l'orchestre ; il permet à l'application hôte de lire et d'écrire directement dans des variables de l'orchestre, sans avoir à utiliser *chnget* ou *chnset* pour copier les données.

## Syntaxe

```
gival chnexport Sname, imode[, itype, idflt, imin, imax]
```

```
gkval chnexport Sname, imode[, itype, idflt, imin, imax]
```

```
gaval chnexport Sname, imode
```

```
gSval chnexport Sname, imode
```

## Initialisation

*imode* -- somme d'au moins un des nombres suivants : 1 pour entrée et 2 pour sortie.

*itype* (facultatif, 0 par défaut) -- sous-type du canal, seulement pour les canaux de contrôle. Les valeurs possibles sont :

- 0: par défaut / indéfini (*idflt*, *imin* et *imax* sont ignorés)
- 1: seulement des valeurs entières
- 2: échelle linéaire
- 3: échelle exponentielle

*idflt* (facultatif, 0 par défaut) -- valeur par défaut, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être supérieur ou égal à *imin* et inférieur ou égal à *imax*.

*imin* (facultatif, 0 par défaut) -- valeur minimale, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être différent de zéro pour l'échelle exponentielle (*itype* = 3).

*imax* (facultatif, 0 par défaut) -- valeur maximale, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être supérieur à *imin*. Dans le cas d'une échelle exponentielle, il doit également avoir le même signe que *imin*.

## Notes

Les paramètres du canal (*imode*, *itype*, *idflt*, *imin* et *imax*) ne sont que des indications pour l'application hôte ou un logiciel externe accédant au bus par l'API, et ils ne restreignent en rien la lecture ou l'écriture sur le canal.

Bien que la variable globale soit utilisée comme argument de sortie, *chnexport* ne la change pas, et ne s'exécute seulement qu'au taux-i. Si la variable n'a pas été préalablement déclarée, elle est créée par Csound avec une valeur initiale de zéro ou une chaîne de caractères nulle.

## Exemple

L'exemple montre l'utilisation du bus logiciel comme signal de contrôle asynchrone pour fixer la fréquence de coupure du filtre. On suppose qu'un programme externe utilisant l'API fournit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

gkc init 1000 ; set default value
gkc chnexport "cutoff", 1, 3, i(gkc), 500, 2000

instr 1
  a1 oscil p4, p5, 100
  a2 lowpass2 a1, gkc, 200
  out a2
endin
```

## Crédits

Auteur : Istvan Varga  
2005

# chnget

chnget — Lit des données depuis le bus logiciel.

## Description

Lit des données depuis un canal du bus logiciel entrant nommé. Cela implique la déclaration du canal avec *imode=1* (voir aussi *chn\_k*, *chn\_a* et *chn\_S*).

## Syntaxe

```
ival chnget Sname
```

```
kval chnget Sname
```

```
aval chnget Sname
```

```
Sval chnget Sname
```

## Initialisation

*Sname* -- une chaîne de caractères identifiant le canal du bus logiciel nommé à lire.

*ival* -- la valeur de contrôle lue à l'initialisation.

*Sval* -- la chaîne de caractères lue à l'initialisation.

## Exécution

*kval* -- la valeur de contrôle lue pendant l'exécution.

*aval* -- le signal audio lu pendant l'exécution.

## Exemple

L'exemple montre l'utilisation du bus logiciel comme signal de contrôle asynchrone pour fixer la fréquence de coupure du filtre. On suppose qu'un programme externe utilisant l'API fournit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

instr 1
  kc chnget "cutoff"
  a1 oscil p4, p5, 100
  a2 lowpass2 a1, kc, 200
  out a2
endin
```

## Crédits



Auteur : Istvan Varga  
2005

# chnmix

chnmix — Ecrit des données audio vers le bus logiciel nommé, en les mélangeant à la sortie précédente.

## Description

Ajoute un signal audio à un canal du bus logiciel nommé. Cela implique la déclaration du canal avec *imode=2* (voir aussi *chn\_a*).

## Syntaxe

```
chnmix aval, Sname
```

## Initialisation

*Sname* -- une chaîne de caractères indiquant le canal nommé du bus logiciel sur lequel écrire.

## Exécution

*aval* -- le signal audio à écrire pendant l'exécution.

## Crédits

Auteur : Istvan Varga  
2006

# chnparams

chnparams — Demande les paramètres d'un canal.

## Description

Demande les paramètres d'un canal (s'il n'existe pas, toutes les valeurs retournées sont nulles).

## Syntaxe

*itype*, *imode*, *ictltype*, *idflt*, *imin*, *imax* **chnparams**

## Initialisation

*itype* -- type des données du canal (1 : contrôle, 2 : audio, 3 : chaînes de caractères)

*imode* -- somme d'au moins un des nombres suivants : 1 pour entrée et 2 pour sortie.

*ictltype* -- paramètre spécial seulement pour les canaux de contrôle ; s'il n'est pas disponible, il est mis à zéro.

*idflt* -- paramètre spécial seulement pour les canaux de contrôle ; s'il n'est pas disponible, il est mis à zéro.

*imin* -- paramètre spécial seulement pour les canaux de contrôle ; s'il n'est pas disponible, il est mis à zéro.

*imax* -- paramètre spécial seulement pour les canaux de contrôle ; s'il n'est pas disponible, il est mis à zéro.

## Crédits

Auteur : Istvan Varga  
2005

# chnrecv

chnrecv — Recieves data from the software bus.

## Description

Receives data from a channel of the inward named software bus. Implies declaring the channel with imode=1 (see also chn\_k, chn\_a, and chn\_S).

## Syntax

ival **chnrecv** Sname

kval **chnrecv** Sname

aval **chnrecv** Sname

Sval **chnrecv** Sname

## Initialization

*Sname* -- a string that identifies a channel of the named software bus to read.

## Performance

*ival* -- the control value read at i-time.

*kval* -- the control value read at performance time.

*aval* -- the audio signal read at performance time.

*Sval* -- the string value read at i-time.



### Note

## Example

The example shows the software bus being used as an asynchronous control signal to select a filter cutoff. It assumes that an external program that has access to the API is feeding the values.

```
sr = 44100
ksmps = 100
nchnls = 1

instr 1
  kc chnrecv "cutoff"
  a1 oscil p4, p5, 100
  a2 lowpass2 a1, kc, 200
  out a2
endin
```

## Credits

Author: Istvan Varga  
2005

# chnsend

chnsend — Sends data via the named software bus.

## Description

Send to a channel of the named software bus. Implies declaring the channel with imode=2 (see also chn\_k, chn\_a, and chn\_S).

## Syntax

**chnsend** ival, Sname

**chnsend** kval, Sname

**chnsend** aval, Sname

**chnsend** Sval, Sname

## Initialization

*Sname* -- a string that indicates which named channel of the software bus to send to.

## Performance

*ival* -- the control value to write at i-time.

*kval* -- the control value to write at performance time.

*aval* -- the audio signal to write at performance time.

*Sval* -- the string value to write at i-time.

## Example

The example shows the software bus being used to write pitch information to a controlling program.

```
sr = 44100
ksmps = 100
nchnls = 1

instr 1
  al in
  kp,ka pitchamdf al
  chnsend kp, "pitch"
endin
```

## See also

*chnrecv*, *chnset*, *chnget*

## Credits

Author: Istvan Varga  
2005

# chnset

chnset — Ecrit des données vers le bus logiciel nommé.

## Description

Ecrit sur un canal du bus logiciel nommé. Cela implique la déclaration du canal avec *imod=2* (voir aussi *chn\_k*, *chn\_a* et *chn\_S*).

## Syntaxe

```
chnset ival, Sname
```

```
chnset kval, Sname
```

```
chnset aval, Sname
```

```
chnset Sval, Sname
```

## Initialisation

*Sname* -- une chaîne de caractères indiquant le canal nommé du bus logiciel sur lequel écrire.

*ival* -- la valeur de contrôle écrite à l'initialisation.

*Sval* -- la chaîne de caractères écrite à l'initialisation.

## Exécution

*kval* -- la valeur de contrôle écrite pendant l'exécution.

*aval* -- le signal audio écrit pendant l'exécution.

## Exemple

L'exemple montre l'utilisation du bus logiciel pour écrire une information de hauteur vers un programme de contrôle.

```
sr = 44100
kr = 100
ksmps = 1

instr 1
  al in
  kp,ka pitchamdf al
  chnset kp, "pitch"
endin
```

## Crédits



Auteur : Istvan Varga  
2005

# chuap

chuap — Simule un oscillateur de Chua, un oscillateur RLC avec une résistance active, qui peut avoir bifurcation et attracteurs chaotiques, avec un contrôle de taux-k des éléments du circuit.

## Description

Simule un oscillateur de Chua, un oscillateur RLC avec une résistance active, qui peut avoir bifurcation et attracteurs chaotiques, avec un contrôle de taux-k des éléments du circuit.

## Syntaxe

```
aI3, aV2, aV1 chuap kL, kR0, kC1, kG, kGa, kGb, kE, kC2, iI3, iV2, iV1, ktime_step
```

## Initialisation

*iI3* -- Courant initial dans G

*iV2* -- Tension initiale aux bornes de C2

*iV1* -- Tension initiale aux bornes de C1

## Exécution

*kL* -- Inductance L

*kR0* -- Résistance R0

*kC1* -- Capacité C1

*kG* -- Résistance G

*kGa* -- Résistance V (terme non linéaire)

*kGb* -- Résistance V (terme non linéaire)

*kGb* -- Résistance V (terme non linéaire)

*ktime\_step* -- Pas temporel de l'équation aux différences, permet de contrôler plus ou moins la hauteur.

*L'oscillateur de Chua* est un simple oscillateur RLC avec une résistance active. L'oscillateur peut être amené à une bifurcation de période, et ainsi vers le chaos, à cause de la réponse non linéaire de la résistance active.

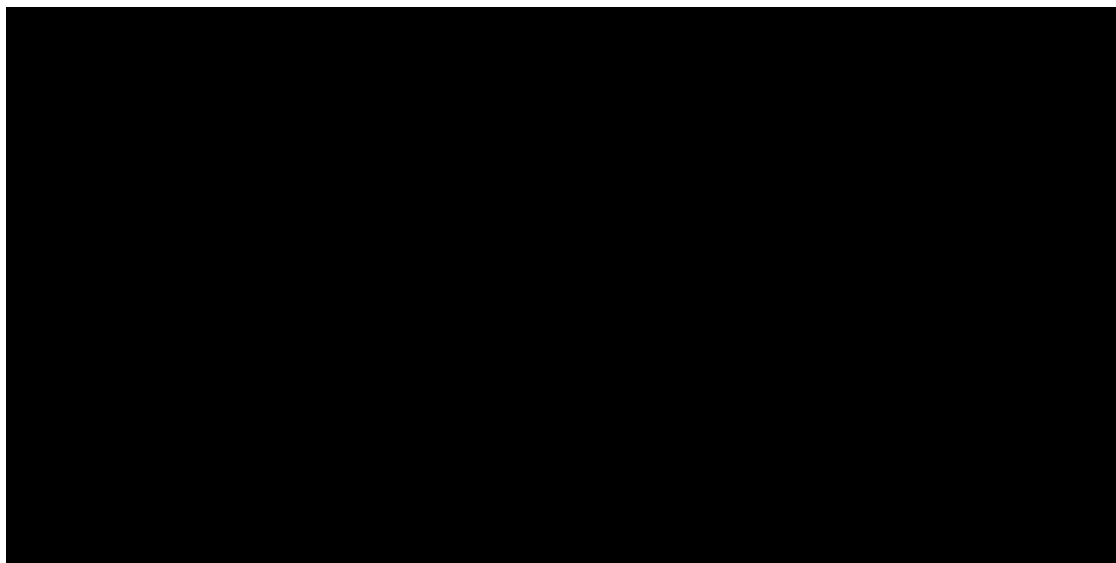


Diagramme du Circuit de l'Oscillateur de Chua

Le circuit est décrit par un ensemble de trois équations différentielles ordinaires appelées équations de Chua :

$$\frac{dI_3}{dt} = -\frac{R_0}{L} I_3 - \frac{1}{L} V_2$$

$$\frac{dV_2}{dt} = -\frac{1}{C_2} I_3 - \frac{G}{C_2} (V_2 - V_1)$$

$$\frac{dV_1}{dt} = \frac{G}{C_1} (V_2 - V_1) - \frac{1}{C_1} f(V_1)$$

où  $f()$  est une fonction dsicontinue par morceaux simulant la résistance active :

$$f(V_1) = G_b V_1 + - (G_a - G_b)(|V_1 + E| - |V_1 - E|)$$

Une solution  $(I_3, V_2, V_1)(t)$  de ces équations partant d'un état initial  $(I_3, V_2, V_1)(0)$  est appelée une trajectoire de l'oscillateur de Chua. L'implémentation dans Csound est une simulation de l'oscillateur de Chua par une équation aux différences avec intégration de Runge-Kutta.



### Note

Cet algorithme utilise des boucles de rétroaction internes non linéaires ce qui fait dépendre le résultat audio du taux d'échantillonnage de l'orchestre. Par exemple, si l'on développe un projet avec  $sr=48000\text{Hz}$  et si l'on veut produire un CD audio de ce projet, il faut enregistrer un fichier avec  $sr=48000\text{Hz}$ , puis sous-échantillonner ce fichier à  $44100\text{Hz}$  avec l'utilitaire *srconv*.



## Avertissement

Attention ! Certains jeux de paramètres produiront des pics d'amplitude ou une rétroaction positive pouvant endommager vos haut-parleurs.

## Exemples

Voici un exemple de l'opcode chuap. Il utilise le fichier *chuap.csd* [examples/chuap.csd].

### Exemple 79. Exemple de l'opcode chuap.

```
<CsoundSynthesizer>
<CsOptions>
csound -RWfo chuas_oscillator.wav
</CsOptions>
<CsInstruments>
sr          =          44100
ksmps       =          100
nchnls      =          2
0dbfs       =          10000

gibuzztable  ftgen      1, 0, 16384, 10, 1

instr 1
; sys_variables = system_vars(5:12); % L,R0,C2,G,Ga,Gb,E,C1 or p8:p15
; integ_variables = [system_vars(14:16),system_vars(1:2)]; % x0,y0,z0,dataset_size,step
istep_size   =          p5
iL           =          p8
iR0          =          p9
iC2          =          p10
iG           =          p11
iGa          =          p12
iGb          =          p13
iE           =          p14
iC1          =          p15
iI3          =          p17
iV2          =          p18
iV1          =          p19
iattack      =          0.02
isustain     =          p3
irelease     =          0.02
p3           =          iattack + isustain + irelease
iscale       =          1.0
adamping     =          0.0, iattack, iscale, isustain, iscale, irelease, 0.0
buzz         =          5000, 440, sr/440, gibuzztable
aguide       =          chuap    iL, iR0, iC2, iG, iGa, iGb, iE, iC1, iI3, iV2, iV1, istep_size
aI3, aV2, aV1 =          balance aV2, aguide
asignal      =          outs    adamping * asignal, adamping * asignal
endin

</CsInstruments>
<CsScore>
;          Adapted from ABC++ MATLAB example data.
i 1 0 20 1500 .1 -1 -1 -0.00707925 0.00001647 100 1 -.99955324 -1.00028375 1 -.00222159 204.8 -2.362
i 1 + 20 1500 .425 0 -1 1.3506168 0 -4.50746268737 -1 2.4924 .93 1 1 0 -22.28662665 .00
i 1 + 20 1024 .05 -1 -1 0.00667 0.000651 10 -1 .856 1.1 1 .06 51.2 -20.200590133667 .172539323
i 1 + 20 1024 0.05 -1 -1 0.00667 0.000651 10 -1 0.856 1.1 1 0.1 153.6 21.12496758 0.03001749 0.5158286
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Inventeur de l'oscillateur de Chua : *Leon O. Chua* [<http://www.eecs.berkeley.edu/~chua>]

Auteur de la simulation dans MATLAB : James Patrick McEvoy *MATLAB Adventures in Bifurcations and Chaos* (ABC++)

<http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=3541>

Auteur du portage dans Csound : Michael Gogins

Nouveau dans la version 5.09 de Csound

Note ajoutée par François Pinot, août 2009

# cigoto

cigoto — Transfert conditionnel du contrôle pendant la phase d'initialisation.

## Description

Tranfert conditionnel du contrôle vers l'instruction étiquetée par *label*, lors de la phase d'initialisation seulement.

## Syntaxe

**cigoto** condition, label

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression, et où *condition* utilise un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exemples

Voici un exemple de l'opcode cigoto. Il utilise le fichier *cigoto.csd* [examples/cigoto.csd].

### Exemple 80. Exemple de l'opcode cigoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc            ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the value of the 4th p-field from the score.
iparam = p4

; If iparam is 1 then play the high note.
; If not then play the low note.
cigoto (iparam ==1), highnote
igoto lownote

highnote:
ifreq = 880
goto playit

lownote:
ifreq = 440
goto playit

playit:
```

```

; Print the values of iparam and ifreq.
print iparam
print ifreq

a1 oscil 10000, ifreq, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

instr 1: iparam = 0.000
instr 1: ifreq = 440.000
instr 1: iparam = 1.000
instr 1: ifreq = 880.000

```

## Voir Aussi

*cggoto, ckgoto, cngoto, goto, if, kgoto, rigoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

# ckgoto

ckgoto — Transfert conditionnel du contrôle lors des phases d'exécution.

## Description

Tranfert conditionnel du contrôle vers l'instruction étiquetée par *label*, lors des phases d'exécution seulement.

## Syntaxe

**ckgoto** condition, label

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression, et où *condition* utilise un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exemples

Voici un exemple de l'opcode ckgoto. Il utilise le fichier *ckgoto.csd* [examples/ckgoto.csd].

### Exemple 81. Exemple de l'opcode ckgoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d            ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ckgoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Change kval linearly from 0 to 2 over
; the period set by the third p-field.
kval line 0, p3, 2

; If kval is greater than or equal to 1 then play the high note.
; If not then play the low note.
ckgoto (kval >= 1), highnote
      kgoto lownote

highnote:
  kfreq = 880
  goto playit

lownote:
  kfreq = 440
  goto playit
```



```
playit:
; Print the values of kval and kfreq.
printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

al oscil 10000, kfreq, 1
out al
endin

</CsInstruments>
<CsScore>

; Table: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
kval = 1.999639, kfreq = 880.000000
```

## Voir Aussi

*cggoto, cigoto, cngoto, goto, if, igoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

# clear

clear — Met à zéro une liste de signaux audio.

## Description

*clear* met à zéro une liste de signaux audio.

## Syntaxe

```
clear avar1 [, avar2] [, avar3] [...]
```

## Exécution

*avar1*, *avar2*, *avar3*, ... -- signals to be zeroed

*clear* met à zéro chaque échantillon de chacun des signaux audio donnés lors de son exécution. C'est comme si l'on écrivait  $avarN = 0$  dans l'orchestre pour chaque variable spécifiée. Typiquement, *clear* est utilisé avec des variables globales qui combinent plusieurs signaux venant de sources différentes et qui sont changées à chaque passe au taux-k (boucle d'exécution) par toutes les instances d'instrument actives. Après la dernière utilisation d'une de ces variables et avant la passe de taux-k suivante, il faut l'effacer afin qu'elle n'ajoute pas les signaux du cycle suivant au précédent résultat. *clear* est particulièrement utile en combinaison avec *vincr* (incrément de variable) lors de leur utilisation conjointe avec des opcodes de sortie dans un fichier comme *fout*.

## Exemples

Voir l'exemple de l'opcode *fout*.

## Voir Aussi

*vincr*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# clfilt

clfilt — Implements low-pass and high-pass filters of different styles.

## Description

Implements the classical standard analog filter types: low-pass and high-pass. They are implemented with the four classical kinds of filters: Butterworth, Chebyshev Type I, Chebyshev Type II, and Elliptical. The number of poles may be any even number from 2 to 80.

## Syntax

```
ares clfilt asig, kfreq, itype, inpol [, ikind] [, ipbr] [, isba] [, iskip]
```

## Initialization

*itype* -- 0 for low-pass, 1 for high-pass.

*inpol* -- The number of poles in the filter. It must be an even number from 2 to 80.

*ikind* (optional) -- 0 for Butterworth, 1 for Chebyshev Type I, 2 for Chebyshev Type II, 3 for Elliptical. Defaults to 0 (Butterworth)

*ipbr* (optional) -- The pass-band ripple in dB. Must be greater than 0. It is ignored by Butterworth and Chebyshev Type II. The default is 1 dB.

*isba* (optional) -- The stop-band attenuation in dB. Must be less than 0. It is ignored by Butterworth and Chebyshev Type I. The default is -60 dB.

*iskip* (optional) -- 0 initializes all filter internal states to 0. 1 skips initialization. The default is 0.

## Performance

*asig* -- The input audio signal.

*kfreq* -- The corner frequency for low-pass or high-pass.

## Examples

Here is an example of the clfilt opcode as a low-pass filter. It uses the file *clfilt\_lowpass.csd* [examples/clfilt\_lowpass.csd].

### Exemple 82. Example of the clfilt opcode as a low-pass filter.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```

```

-odac          -iadc      -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o clfilt_lowpass.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1
; White noise signal
asig rand 22050

out asig
endin

; Instrument #2 - a filtered noise waveform.
instr 2
; White noise signal
asig rand 22050

; Lowpass filter signal asig with a
; 10-pole Butterworth at 500 Hz.
al clfilt asig, 500, 0, 10

out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

Here is an example of the `clfilt` opcode as a high-pass filter. It uses the file `clfilt_highpass.csd` [examples/clfilt\_highpass.csd].

### Exemple 83. Example of the `clfilt` opcode as a high-pass filter.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac          -iadc      -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o clfilt_highpass.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1 - an unfiltered noise waveform.
instr 1

```

```
; White noise signal
asig rand 22050

out asig
endin

; Instrument #2 - a filtered noise waveform.
instr 2
; White noise signal
asig rand 22050

; Highpass filter signal asig with a 6-pole Chebyshev
; Type I at 20 Hz with 3 dB of passband ripple.
al clfilt asig, 20, 1, 6, 1, 3

out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Erik Spjut

New in version 4.20

# clip

clip — Rogne un signal à une limite prédéfinie.

## Description

Coupe un signal de taux-*a* à une limite prédéfinie, de manière « douce », en utilisant une méthode choisie parmi les trois possibles.

## Syntaxe

```
ares clip asig, imeth, ilimit [, iarg]
```

## Initialisation

*imeth* -- choisit la méthode de coupure. La valeur par défaut est 0. Les méthodes sont :

- 0 = méthode de Bram de Jong (par défaut)
- 1 = coupure par sinus
- 2 = coupure par tanh

*ilimit* -- valeur limite

*iarg* (facultatif, 0.5 par défaut) -- lorsque *imeth* = 0, indique le point, compris entre 0 et 1, où la coupure commence. N'est pas utilisé si *imeth* = 1 ou 2. Sa valeur par défaut est 0.5.

## Exécution

*asig* -- signal de taux-*a* en entrée

La méthode de Bram de Jong (*imeth* = 0) applique l'algorithme (en notant *ilimit* comme *limit* et *iarg* comme *a* :

```
|x| >= 0 and |x| <= (limit*a): f(x) = f(x)
|x| > (limit*a) and |x| <= limit: f(x) = sign(x) * (limit*a+(x-limit*a)/(1+((x-limit*a)/(limit*(1-a))))
|x| > limit: f(x) = sign(x) * (limit*(1+a))/2
```

La seconde méthode (*imeth* = 1) est la coupure par sinus :

```
|x| < limit: f(x) = limit * sin(π*x/(2*limit)), |x| >= limit: f(x) = limit * sign(x)
```

La troisième méthode (*imeth* = 2) est la coupure par tanh :

$|x| < limit: f(x) = limit * \tanh(x/limit)/\tanh(1), \quad |x| \geq limit: f(x) = limit * \text{sign}(x)$



## Note

Il semble que la méthode 1 n'était pas fonctionnelle dans la version 4.07 de Csound.

## Exemples

Voici un exemple de l'opcode clip. Il utilise le fichier *clip.csd* [examples/clip.csd].

### Exemple 84. Exemple de l'opcode clip.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out      Audio in      No messages
-odac            -iadc          -d            ;;RT audio I/O
; -o clip.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a noisy waveform.
arnd rand 44100
; Clip the noisy waveform's amplitude to 20,000
a1 clip arnd, 2, 20000

out a1
endin

; Instrument #2.
instr 2
; Generate a noisy waveform.
arnd rand 44100
; Clip the noisy waveform's amplitude to 10,000
a1 clip arnd, 2, 10000

out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist Ltd.  
Bath, UK  
Août, 2000

Nouveau dans la version 4.07 de Csound

Septembre 2009 : grâce à une note de Paolo Dell'Oso, les formules ont été corrigées.



# clock

clock — Obsolète.

## Description

Obsolète. Utiliser plutôt l'opcode *rtclock*.

# clockoff

clockoff — Arrête l'une des horloges internes.

## Description

Arrête l'une des horloges internes.

## Syntaxe

`clockoff inum`

## Initialisation

*inum* -- le numéro d'une horloge. Il y a 32 horloges numérotées de 0 à 31. Toutes les autres valeurs correspondent à l'horloge numéro 32.

## Exécution

Entre deux opcodes *clockon* et *clockoff*, le temps CPU utilisé est accumulé dans l'horloge. La précision dépend de la machine et elle est de l'ordre de la milliseconde sur les systèmes UNIX et Windows. L'opcode *readclock* lit la valeur courante d'une horloge pendant une phase d'initialisation.

## Exemples

Voir l'exemple de l'opcode *readclock*.

## Voir Aussi

*clockon*, *readclock*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Juillet 1999

Nouveau dans la version 3.56 de Csound

# clockon

clockon — Démarre l'une des horloges internes.

## Description

Démarre l'une des horloges internes.

## Syntaxe

`clockon inum`

## Initialisation

*inum* -- le numéro d'une horloge. Il y a 32 horloges numérotées de 0 à 31. Toutes les autres valeurs correspondent à l'horloge numéro 32.

## Exécution

Entre deux opcodes *clockon* et *clockoff*, le temps CPU utilisé est accumulé dans l'horloge. La précision dépend de la machine et elle est de l'ordre de la milliseconde sur les systèmes UNIX et Windows. L'opcode *readclock* lit la valeur courante d'une horloge pendant une phase d'initialisation.

## Exemples

Voir l'exemple de l'opcode *readclock*.

## Voir Aussi

*clockoff*, *readclock*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Juillet 1999

Nouveau dans la version 3.56 de Csound

# cngoto

cngoto — Transfère le contrôle à chaque passage si la condition n'est pas vraie.

## Description

Transfère le contrôle à chaque passage si la condition n'est *pas* vraie.

## Syntaxe

```
cngoto condition, label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression, et où *condition* utilise un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exemples

Voici un exemple de l'opcode cngoto. Il utilise le fichier *cngoto.csd* [examples/cngoto.csd].

### Exemple 85. Exemple de l'opcode cngoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; -o cngoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Change kval linearly from 0 to 2 over
; the period set by the third p-field.
kval line 0, p3, 2

; If kval *is not* greater than or equal to 1 then play
; the high note. Otherwise, play the low note.
cngoto (kval >= 1), highnote
      kgoto lownote

highnote:
  kfreq = 880
  goto playit

lownote:
  kfreq = 440
  goto playit

playit:
; Print the values of kval and kfreq.
printks "kval = %f, kfreq = %f\n", 1, kval, kfreq
```

```
    al oscil 10000, kfreq, 1
    out al
endin

</CsInstruments>
<CsScore>

; Table: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
kval = 0.000000, kfreq = 880.000000
kval = 0.999732, kfreq = 880.000000
kval = 1.999639, kfreq = 440.000000
```

## Voir Aussi

*cggoto, cigoto, ckgoto, goto, if, igoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.21

# comb

comb — Reverberates an input signal with a « colored » frequency response.

## Description

Reverberates an input signal with a « colored » frequency response.

## Syntax

```
ares comb asig, krvt, ilpt [, iskip] [, insmps]
```

## Initialization

*ilpt* -- loop time in seconds, which determines the « echo density » of the reverberation. This in turn characterizes the « color » of the *comb* filter whose frequency response curve will contain *ilpt* \* *sr*/2 peaks spaced evenly between 0 and *sr*/2 (the Nyquist frequency). Loop time can be as large as available memory will permit. The space required for an *n* second loop is  $4n*sr$  bytes. Delay space is allocated and returned as in *delay*.

*iskip* (optional, default=0) -- initial disposition of delay-loop data space (cf. *reson*). The default value is 0.

*insmps* (optional, default=0) -- delay amount, as a number of samples.

## Performance

*krvt* -- the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude).

This filter reiterates input with an echo density determined by loop time *ilpt*. The attenuation rate is independent and is determined by *krvt*, the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude). Output from a comb filter will appear only after *ilpt* seconds.

## Examples

Here is an example of the comb opcode. It uses the file *comb.csd* [examples/comb.csd].

### Exemple 86. Example of the comb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o comb.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the audio mixer.
gamix init 0

; Instrument #1.
instr 1
; Generate a source signal.
a1 oscili 30000, cpspch(p4), 1
; Output the direct sound.
out a1

; Add the source signal to the audio mixer.
gamix = gamix + a1
endin

; Instrument #99 (highest instr number executed last)
instr 99
krvt = 1.5
ilpt = 0.1

; Comb-filter the mixed signal.
a99 comb gamix, krvt, ilpt
; Output the result.
out a99

; Empty the mixer for the next pass.
gamix = 0
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; p4 = frequency (in a pitch-class)
; Play Instrument #1 for a tenth of a second, p4=7.00
i 1 0 0.1 7.00
; Play Instrument #1 for a tenth of a second, p4=7.02
i 1 1 0.1 7.02
; Play Instrument #1 for a tenth of a second, p4=7.04
i 1 2 0.1 7.04
; Play Instrument #1 for a tenth of a second, p4=7.06
i 1 3 0.1 7.06

; Make sure the comb-filter remains active.
i 99 0 5
e

</CsScore>
</CsoundSynthesizer>

```

## See Also

*alpass*, *reverb*, *valpass*, *vcomb*

## Credits

Author: William « Pete » Moss (*vcomb* and *valpass*)  
 University of Texas at Austin  
 Austin, Texas USA  
 January 2002

Example written by Kevin Conder.



# compress

compress — Compresse, limite, dilate, atténue ou impose un seuil à un signal audio.

## Description

Cette unité fonctionne comme un compresseur audio, un limiteur, un expander ou un noise gate, avec un coude arrondi ou non et des caractéristiques d'exécution variant dynamiquement. Il prend deux signaux audio en entrée, *aasig* et *acsig*, le premier étant modifié par l'analyse courante du second. Les deux signaux peuvent être le même signal, ou le premier peut être modifié par un signal de contrôle différent.

**compress** examine d'abord le signal de contrôle *acsig* en faisant une détection d'enveloppe. Celle-ci est déterminée par deux valeurs de contrôle *katt* et *krel*, définissant les constantes d'attaque et de relachement (en secondes) du détecteur. Le détecteur suit les crêtes (pas la valeur efficace) du signal de contrôle. Les valeurs typiques sont 0.01 et 0.1, la dernière étant habituellement du même ordre que *ilook*.

L'enveloppe courante est alors convertie en décibels puis passe par une fonction de sélection pour déterminer quelle action du compresseur (s'il y en a une) doit être appliquée. La fonction de sélection est définie par quatre valeurs de contrôle en décibels. Elles sont données sous forme de valeurs positives, où 0 dB correspond à une amplitude de 1, et 90 dB correspond à une amplitude de 32768.

## Syntaxe

ar **compress** aasig, acsig, kthresh, kloknee, khiknee, kratio, katt, krel, ilook

## Initialisation

*ilook* -- temps de prospection en secondes, pendant lequel un déclenchement d'enveloppe interne peut détecter ce qui se passe. Cela induit un délai entre l'entrée et la sortie, mais une petite durée de prospection améliore les performances du détecteur d'enveloppe. 0.05 secondes est une valeur typique, suffisante pour détecter les crêtes de la fréquence la plus basse dans *acsig*.

## Exécution

*kthresh* -- fixe le niveau le plus bas en décibels qui sera autorisé à traverser le module. Normalement 0 ou moins, mais si le seuil est plus élevé, les signaux de basse énergie tel que le bruit de fond commenceront à être enlevés.

*kloknee*, *khiknee* -- coude de la courbe en décibels indiquant où commencent la compression ou l'expansion. Cela fixe les limites d'un coude arrondi joignant la ligne 1:1 des basses amplitudes et la ligne du rapport de compression des fortes amplitudes. 48 et 60 dB sont des valeurs typiques. Si les deux points sont égaux, le coude est anguleux.

*kratio* -- rapport de compression lorsque le signal est au-delà du coude. La valeur 2 renforce la sortie d'un décibel pour chaque doublement du gain en entrée ; 3 renforce de un pour trois ; 20 de un pour vingt, etc. Les rapports inverses provoquent une expansion du signal : 0.5 donne deux pour un, 0.25 quatre pour un, etc. La valeur 1 ne provoque aucun changement.

Les actions de *compress* dépendent du réglage des paramètres. Un compresseur-limiteur à coude anguleux, par exemple, est obtenu avec une attaque proche de zéro, des limites de coude égales, et un rapport très élevé (disons 100). Un noise-gate plus expander est obtenu avec un seuil positif et un rapport fractionnaire au-delà du coude. Un compresseur de musique déclenché par la voix (ducker) est obtenu en af-

fectant la musique à *aasig* et la voix à *acsig*. Un de-esser de voix est obtenu en affectant la voix au deux, avec un filtre passe-bande précédant l'entrée *acsig* pour renforcer les sifflantes. Il est nécessaire d'expérimenter chaque application pour trouver les meilleurs réglages des paramètres ; ceux-ci sont de taux-k pour faciliter cette expérimentation.

## Exemples

```
aout compress amus, avoc, 0, 40, 60, 3, .1, .5, .02 ; voice-activated compressor  
                                         ; with low-level sensitivity
```

## Crédits

Ecrit par Barry L. Vercoe pour Extended Csound et introduit dans Csound 5.02.

# connect

`connect` — Connects the named outlet ports of source instruments to the named inlet ports of sink instruments.

## Description

The `connect` opcode, valid only in orchestra headers, sends the signals from the indicated outlet in all instances of the indicated source instrument to the indicated inlet in all instances of the indicated sink instrument. Each inlet instance receives the sum of the signals in all outlet instances. Thus multiple instances of an outlet may fan in to one instance of an inlet, or one instance of an outlet may fan out to multiple instances of an inlet.

## Syntax

```
connect Tsource1, Soutlet1, Tsink1, Sinlet1
```

## Initialization

*Tsource1* -- String name, or number, of the source instrument.

*Soutlet1* -- String name of an outlet port in the source instrument.

*Tsink1* -- String name, or number, of the sink instrument.

*Sinlet1* -- String name of an inlet port in the sink instrument.

## Performance

During performance, the signals in all named outlets are summed in the named inlets to which they have been connected using the `connect` opcode. When new instances of instruments are allocated during performance, their inlets, outlets, and connections also are copied.



### Order is important

The order of definition of instruments is important. A source instrument must be defined in the orchestra before any of the sink instruments to which its outlets are connected.

## See Also

*outleta outletk outletf inleta inletk inletf alwayson fngenonce*

## Credits

By: Michael Gogins 2009

# control

control — Configurable slider controls for realtime user input.

## Description

Configurable slider controls for realtime user input. Requires Winsound or TCL/TK. *control* reads a slider's value.

## Syntax

```
kres control knum
```

## Performance

*knum* -- number of the slider to be read.

Calling *control* will create a new slider on the screen. There is no theoretical limit to the number of sliders. Windows and TCL/TK use only integers for slider values, so the values may need rescaling. GUIs usually pass values at a fairly slow rate, so it may be advisable to pass the output of control through *port*.

## Examples

See the *setctrl* opcode for an example.

## See Also

*setctrl*

## Credits

Author: John ffitch  
University of Bath, Codemist. Ltd.  
Bath, UK  
May, 2000

New in Csound version 4.06

## convle

convle — Identique à l'opcode *convolve*.

## Description

Identique à l'opcode *convolve*.

# convolve

convolve — Convolution d'un signal par une réponse impulsionnelle.

## Description

La sortie est le produit de convolution du signal *ain* par la réponse impulsionnelle contenue dans *ifilcod*. S'il y a plus d'un signal de sortie, chacun sera obtenu par convolution avec la même réponse impulsionnelle. Noter qu'il est considérablement plus efficace d'utiliser une instance de l'opérateur lorsque l'on traite une entrée mono pour créer des sorties stéréo ou quadraphoniques.

Note : cet opcode peut aussi s'écrire *convle*.

## Syntaxe

```
ar1 [, ar2] [, ar3] [, ar4] convolve ain, ifilcod [, ichannel]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères définissant un fichier de données contenant une réponse impulsionnelle. Un entier définit le suffixe d'un fichier *convolve.m* ; une chaîne de caractères (entre guillemets) donne un nom de fichier, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SADIR (si elle est définie). Le fichier de données contient la transformée de Fourier d'une réponse impulsionnelle. L'occupation mémoire dépend de la taille du fichier de données qui est lu en entier et gardé en mémoire durant le calcul, mais qui est partagé par des appels multiples.

*ichannel* (facultatif) -- quel canal du fichier de données de la réponse impulsionnelle utiliser.

## Exécution

*ain* -- signal audio en entrée.

*convolve* implémente la convolution rapide. Le sortie de cet opérateur est retardée en fonction de l'entrée. On peut calculer le délai avec les formules suivantes :

```
Pour (1/kr) <= IRdur:
    Delay = ceil(IRdur * kr) / kr
Pour (1/kr) > IRdur:
    Delay = IRdur * ceil(1/(kr*IRdur))
Où :
    kr = taux de contrôle de Csound
    IRdur = durée, en secondes, de la réponse impulsionnelle
    ceil(n) = le plus petit entier qui n'est pas inférieur à n
```

Il faut également faire attention à prendre en compte le délai initial, s'il y en a un, de la réponse impulsionnelle. Par exemple, si une réponse impulsionnelle est créée à partir d'un enregistrement, le fichier son peut ne pas avoir de délai initial. Il faut ainsi soit s'assurer que le fichier son a la quantité correcte de zéro de remplissage au début, soit, de préférence compenser ce retard dans l'orchestre (cette dernière méthode étant plus efficace). Pour compenser le délai dans l'orchestre, il faut soustraire le délai initial du résultat calculé au moyen des formules ci-dessus, lorsque l'on calcule le délai requis à introduire dans la

pas audio non "réverbérée".

Pour des applications typiques telles que la réverbération, le délai sera de l'ordre de 0.5 à 1.5 secondes, ou même plus long. Cela rend cette implémentation impropre aux applications en temps réel. Il est cependant concevable de l'utiliser pour du filtrage en temps réel, si le nombre de points de lecture est suffisamment petit.

L'auteur a l'intention de créer un opérateur de plus haut niveau qui mélangera le signal original et le signal réverbéré, en utilisant automatiquement la bonne quantité de délai.

## Exemples

Créer le fichier de réponse impulsionnelle dans le domaine fréquentiel au moyen de l'utilitaire *cvanal utility* :

```
csound -Ucvanal ll_44.wav ll_44.cv
```

Déterminer la durée de la réponse impulsionnelle. Pour une grande précision, déterminer le nombre de trames d'échantillon dans le fichier de la réponse impulsionnelle, puis calculer la durée avec :

$\text{durée} = (\text{trames d'échantillons}) / (\text{taux d'échantillonnage du fichier son})$

Cela est dû au fait que l'utilitaire *sndinfo* ne fournit la durée arrondie qu'au 10 ms les plus proches. Si l'on dispose d'un utilitaire qui fournit la durée avec la précision requise, alors il suffit d'utiliser directement la valeur retournée.

```
sndinfo ll_44.wav
```

length = 60822 samples, sample rate = 44100

Duration = 60822/44100 = 1.379s.

Déterminer le délai initial, s'il existe, de la réponse impulsionnelle. Si le délai initial de la réponse impulsionnelle n'a pas été enlevé, alors on peut ignorer cette étape. S'il a été enlevé, la seule manière de connaître le délai initial est de se procurer l'information séparément. Pour cet exemple, on suppose que le délai initial est de 60 ms (0.06 s).

Déterminer le délai qu'il faut nécessairement appliqué au signal original pour l'aligner sur le signal convolué :

Si  $kr = 441$ :

$1/kr = 0.0023$ , qui est  $\leq IR_{dur}$  (1.379s), ainsi :

$\text{Delay1} = \text{ceil}(IR_{dur} * kr) / kr$   
 $= \text{ceil}(608.14) / 441$   
 $= 609/441$   
 $= 1.38s$

En prenant comme délai initial :

$\text{Delay2} = 0.06s$

Total delay = delay1 - delay2  
 = 1.38 - 0.06  
 = 1.32s

Créer un fichier .orc, par exemple :

```

; Simple demonstration of CONVOLVE operator, to apply reverb.
sr = 44100
kr = 441
ksmps = 100
nchnls = 2
instr 1
imix = 0.22 ; Wet/dry mix. Vary as desired.
; NB: 'Small' reverbs often require a much higher
; percentage of wet signal to sound interesting. 'Large'
; reverbs seem require less. Experiment! The wet/dry mix is
; very important - a small change can make a large difference.
ivol = 0.9 ; Overall volume level of reverb. May need to adjust
; when wet/dry mix is changed, to avoid clipping.
idel = 1.32 ; Required delay to align dry audio with output of convolve.
; This can be automatically calculated within the orc file,
; if desired.
adry soundin "anechoic.wav" ; input (dry) audio
awet1,awet2 convolve adry,"l1_44.cv" ; stereo convolved (wet) audio
adrydel delay (1-imix)*adry,idel ; Delay dry signal, to align it with
; convolved signal. Apply level
; adjustment here too.
outs ivol*(adrydel+imix*awet1),ivol*(adrydel+imix*awet2)
; Mix wet & dry signals, and output
endin
  
```

## Voir Aussi

*pconvolve, dconv, cvanal.*

## Crédits

Auteur : Greg Sullivan

1996

Nouveau dans la version 3.28



# COS

cos — Calcule une fonction cosinus.

## Description

Retourne cosinus de  $x$  ( $x$  en radians).

## Syntaxe

`cos(x)` (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode cos. Il utilise le fichier *cos.csd* [examples/cos.csd].

### Exemple 87. Exemple de l'opcode cos.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cos.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 25
  i1 = cos(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.991
```

## Voir Aussi

*cosh, cosinv, sin, sinh, sininv, tan, tanh, taninv*

## Crédits

Exemple écrit par Kevin Conder.

# cosh

cosh — Calcule une fonction cosinus hyperbolique.

## Description

Retourne cosinus hyperbolique de  $x$  ( $x$  en radians).

## Syntaxe

`cosh(x)` (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode cosh. Il utilise le fichier *cosh.csd* [examples/cosh.csd].

### Exemple 88. Exemple de l'opcode cosh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cosh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 1
  i1 = cosh(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsSoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.543
```

## Voir Aussi

*cos, cosinv, sin, sinh, sininv, tan, tanh, taninv*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# cosinv

cosinv — Calcule une fonction arccosinus.

## Description

Retourne arccosinus de  $x$  ( $x$  en radians).

## Syntaxe

`cosinv(x)` (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode cosinv. Il utilise le fichier *cosinv.csd* [examples/cosinv.csd].

### Exemple 89. Exemple de l'opcode cosinv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cosinv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = cosinv(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.047
```

## Voir Aussi

*cos, cosh, sin, sinh, sininv, tan, tanh, taninv*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

# cps2pch

`cps2pch` — Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de l'octave.

## Description

Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de l'octave.

## Syntaxe

```
icps cps2pch ipch, iequal
```

## Initialisation

*ipch* -- Nombre en entrée de la forme 8ve.pc, indiquant une octave et quelle note dans l'octave.

*iequal* -- S'il est positif, c'est le nombre d'intervalles égaux de division de l'octave. Doit être inférieur ou égal à 100. S'il est négatif, c'est le numéro d'une table de multiplicateurs de fréquence.



### Note

1. Les lignes suivantes sont équivalentes

```
ia = cpspch(8.02)
ib  cps2pch 8.02, 12
ic  cpsxpch 8.02, 12, 2, 1.02197503906
```

2. C'est un opcode, pas une fonction.
3. Des valeurs négatives pour *ipch* sont permises.

## Exemples

Voici un exemple de l'opcode `cps2pch`. Il utilise le fichier `cps2pch.csd` [examples/cps2pch.csd].

### Exemple 90. Exemple de l'opcode `cps2pch`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```

; -o cps2pch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a normal twelve-tone scale.
ipch = 8.02
iequal = 12

icps cps2pch ipch, iequal

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 293.666
```

Voici un exemple de l'opcode `cps2pch` qui utilise une table de multiplicateurs de fréquence. Il utilise le fichier `cps2pch_ftable.csd` [examples/cps2pch\_ftable.csd].

### Exemple 91. Exemple de l'opcode `cps2pch` qui utilise une table de multiplicateurs de fréquence.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cps2pch_ftable.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
ipch = 8.02

; Use Table #1, a table of frequency multipliers.
icps cps2pch ipch, -1

print icps
endin

</CsInstruments>
<CsScore>

```



```

; Table #1: a table of frequency multipliers.
; Creates a 10-note scale of unequal divisions.
f 1 0 16 -2 1 1.1 1.2 1.3 1.4 1.6 1.7 1.8 1.9

; Play Instrument #1 for one second.
i 1 0 1
e

```

```

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 313.951
```

Voici un exemple de l'opcode `cps2pch` qui utilise une échelle tempérée égale avec 19 divisions de l'octave. Il utilise le fichier `cps2pch_19et.csd` [examples/cps2pch\_19et.csd].

## Exemple 92. Exemple de l'opcode `cps2pch` qui utilise une échelle tempérée égale avec 19 divisions de l'octave.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cps2pch_19et.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use 19ET scale.
ipch = 8.02
iequal = 19

icps cps2pch ipch, iequal

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 281.429
```

## Voir Aussi

*cpspch, cpsxpch*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
1997

Nouveau dans la version 3.492 de Csound

# cpsmidi

cpsmidi — Get the note number of the current MIDI event, expressed in cycles-per-second.

## Description

Get the note number of the current MIDI event, expressed in cycles-per-second.

## Syntax

icps cpsmidi

## Performance

Get the note number of the current MIDI event, expressed in cycles-per-second units, for local processing.



### cpsmidi vs. cpsmidinn

The *cpsmidi* opcode only produces meaningful results in a Midi-activated note (either real-time or from a Midi score with the -F flag). With *cpsmidi*, the Midi note number value is taken from the Midi event that is internally associated with the instrument instance. On the other hand, the *cpsmidinn* opcode may be used in any Csound instrument instance whether it is activated from a Midi event, score event, line event, or from another instrument. The input value for *cpsmidinn* might for example come from a p-field in a textual score or it may have been retrieved from the real-time Midi event that activated the current note using the *notnum* opcode.

## Examples

Here is an example of the cpsmidi opcode. It uses the file *cpsmidi.csd* [examples/cpsmidi.csd].

### Exemple 93. Example of the cpsmidi opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac        -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o cpsmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  i1 cpsmidi

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*aftouch, ampmidi, cpsmidib, cpstmid, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc, cpsmidinn, octmidinn, pchmidinn*

## Credits

Author: Barry L. Vercoe - Mike Berry  
MIT - Mills  
May 1997

Example written by Kevin Conder.

# cpsmidib

cpsmidib — Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in cycles-per-second.

## Description

Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in cycles-per-second.

## Syntax

```
icps cpsmidib [irange]
```

```
kcps cpsmidib [irange]
```

## Initialization

*irange* (optional) -- the pitch bend range in semitones.

## Performance

Get the note number of the current MIDI event, modify it by the current pitch-bend value, and express the result in cycles-per-second units. Available as an i-time value or as a continuous k-rate value.

## Examples

Here is an example of the cpsmidib opcode. It uses the file *cpsmidib.csd* [examples/cpsmidib.csd].

### Exemple 94. Example of the cpsmidib opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac        -iadc      -d           -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o cpsmidib.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 cpsmidib

  print i1
```

```
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*aftouch, ampmidi, cpsmidi, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Credits

Author: Barry L. Vercoe - Mike Berry  
MIT - Mills  
May 1997

Example written by Kevin Conder.

# cpsmidinn

cpsmidinn — Convertit un numéro de note Midi en cycles par seconde.

## Description

Convertit un numéro de note Midi en cycles par seconde.

## Syntaxe

`cpsmidinn` (MidiNoteNumber) (arguments de `taux-i` ou `-k` seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*cpsmidinn* est une fonction qui prend une valeur de `taux-i` ou de `taux-k` représentant un numéro de note Midi et qui retourne la valeur de fréquence équivalente en cycles par seconde (Hertz). Cette conversion suppose que le *do* médian est la note Midi numéro 60 et que le *la* du diapason est accordé à 440 Hz. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.



### cpsmidinn vs. cpsmidi

L'opcode *cpsmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un événement Midi, un événement de partition, un événement en ligne, ou depuis un autre instrument. La valeur d'entrée de *cpsmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode *notnum* à partir de l'évènement Midi en temps-réel qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de `taux-i` ou de `taux-k`. D'un autre côté, l'opcode *cpsmidi* ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option `-F`). Avec *cpsmidi*, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

*cpsmidinn* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 7. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *cpsmidinn*. Il utilise le fichier *cpsmidinn.csd* [exemples/cpsmidinn.csd].

### Exemple 95. Exemple de l'opcode *cpsmidinn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
; i-time loop to print conversion table
imidiNN = 0
loop1:
  icps = cpsmidinn(imidiNN)
  ioct = octmidinn(imidiNN)
  ipch = pchmidinn(imidiNN)

  print imidiNN, icps, ioct, ipch

  imidiNN = imidiNN + 1
if (imidiNN < 128) igoto loop1
```



```

endin

instr 2
; test k-rate converters
kMiddleC = 60
kcps = cpsmidinn(kMiddleC)
koct = octmidinn(kMiddleC)
kpch = pchmidinn(kMiddleC)

printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*octmidinn, pchmidinn, cpsmidi, notnum, cpspch, cpsoct, octcps, octpch, pchoct*

## Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07

# cpsoct

cpsoct — Convertit une valeur octave-point-partie-décimale en cycles par seconde.

## Description

Convertit une valeur octave-point-partie-décimale en cycles par seconde.

## Syntaxe

`cpsoct (oct) (pas de restriction de taux)`

où l'argument entre parenthèses peut être une expression.

## Exécution

*cpsoct* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 8. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + *k1*) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque *k*-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *cpsoct*. Il utilise le fichier *cpsoct.csd* [examples/cpsoct.csd].

### Exemple 96. Exemple de l'opcode *cpsoct*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cpsoct.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Convert an octave-point-decimal value into a
; cycles-per-second value.
ioct = 8.75
icps = cpsoct(ioct)

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 440.000
```

## Voir Aussi

*cspch, octcps, octpch, pchoct, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Exemple écrit par Kevin Conder.

# cpspch

cpspch — Convertit une valeur de classe de hauteur en cycles par seconde.

## Description

Convertit une valeur de classe de hauteur en cycles par seconde.

## Syntaxe

**cpspch** (pch) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*cpsoct* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 9. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Si vous avez besoin de plus de précision de calcul, utilisez plutôt *cps2pch* ou *cpsxpch*.

## Exemples

Voici un exemple de l'opcode *cpspch*. Il utilise le fichier *cpspch.csd* [examples/cpspch.csd].

### Exemple 97. Exemple de l'opcode *cpspch*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpspch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Convert a pitch-class value into a
; cycles-per-second value.
ipch = 8.09
icps = cpspch(ipch)

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 440.000
```

## Voir Aussi

*cps2pch, cpsoct, cpsxpch, octcps, octpch, pchoct, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Exemple écrit par Kevin Conder.

# cpstmid

cpstmid — Get a MIDI note number (allows customized micro-tuning scales).

## Description

This unit is similar to *cpsmidi*, but allows fully customized micro-tuning scales.

## Syntax

```
icps cpstmid ifn
```

## Initialization

*ifn* -- function table containing the parameters (*numgrades*, *interval*, *basefreq*, *basekeymidi*) and the tuning ratios.

## Performance

Init-rate only

*cpsmid* requires five parameters, the first, *ifn*, is the function table number of the tuning ratios, and the other parameters must be stored in the function table itself. The function table *ifn* should be generated by *GEN02*, with normalization inhibited. The first four values stored in this function are:

1. *numgrades* -- the number of grades of the micro-tuning scale
2. *interval* -- the frequency range covered before repeating the grade ratios, for example 2 for one octave, 1.5 for a fifth etc.
3. *basefreq* -- the base frequency of the scale in Hz
4. *basekeymidi* -- the MIDI note number to which *basefreq* is assigned unmodified

After these four values, the user can begin to insert the tuning ratios. For example, for a standard 12 note scale with the base frequency of 261 Hz assigned to the key number 60, the corresponding f-statement in the score to generate the table should be:

```
;      numgrades interval basefreq basekeymidi tuning ratios (equal temp)
f1 0 64 -2 12 2 261 60 1 1.059463094359 1.122462048309 1.189207115003 ..etc...
```

Another example with a 24 note scale with a base frequency of 440 assigned to the key number 48, and a repetition interval of 1.5:

```
;      numgrades interval basefreq basekeymidi tuning-ratios (equal temp)
f1 0 64 -2 24 1.5 440 48 1 1.01 1.02 1.03 ..etc...
```



## Examples

Here is an example of the `cpstmid` opcode. It uses the file `cpstmid.csd` [examples/cpstmid.csd].

### Exemple 98. Example of the `cpstmid` opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc          -d          -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o cpstmid.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
          1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
          1.498, 1.588, 1.682, 1.782, 1.888, 2.000

; Instrument #1.
instr 1
; Use Table #1.
ifn = 1
il cpstmid ifn

print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*cpsmidi*, *GEN02*

## Credits

Author: Gabriel Maldonado  
Italy  
1998

Example written by Kevin Conder.

New in Csound version 3.492

# cpstun

cpstun — Retourne des valeurs d'échelle microtonale au taux-k.

## Description

Retourne des valeurs d'échelle microtonale au taux-k.

## Syntaxe

kcps **cpstun** ktrig, kindex, kfn

## Exécution

*kcps* -- Valeur de retour en cycles par seconde.

*ktrig* -- Un signal utilisé pour déclencher l'évaluation.

*kindex* -- Un nombre entier servant d'indice dans l'échelle.

*kfn* -- Table de fonction contenant les paramètres (numgrades, interval, basefreq, basekeymidi) ainsi que les rapports de hauteur.

Cet opcode est similaire à *cpstmid*, mais son fonctionnement ne nécessite pas le MIDI.

*cpstun* travaille au taux-k. Il permet d'obtenir des échelles microtonales personnalisées. Il nécessite le numéro d'une table de fonction qui contient les rapports de hauteur, et quelques autres paramètres stockés dans la table elle-même.

L'argument *kindex* reçoit des nombres entiers indiquant quel degré de l'échelle donnée doit être converti en Hz. Dans *cpstun*, une nouvelle valeur ne sera évaluée que lorsque *ktrig* contiendra une valeur non nulle. La table de fonction *kfn* sera générée par *GEN02*, les quatre premières valeurs stockées dans la table étant des paramètres qui expriment :

- numgrades -- Le nombre de degrés de l'échelle microtonale.
- interval -- L'intervalle de fréquence couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.
- basefreq -- La fréquence de base de l'échelle en cycles par seconde.
- basekey -- L'indice entier dans l'échelle auquel la fréquence de base sera affectée sans changement.

On peut insérer les rapports de hauteur après ces quatre valeurs. Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 Hz affectée au numéro de touche 60, l'instruction f de la partition pour générer la table sera :

```
;          numgrades  basefreq  tuning-ratios (eq.temp) .....  
;  
f1 0 64 -2 12      2      261    60    1    1.059463 1.12246 1.18920 ..etc...
```

Un autre exemple avec une échelle de 24 degrés et une fréquence de base de 440 affectée au numéro de touche 48, et un intervalle de répétition de 1,5 :

```
;
;          numgrades      basefreq      tuning-ratios .....
;          interval      basekey
f1 0 64 -2      24      1.5      440      48      1      1.01 1.02 1.03 ..etc...
```

## Exemples

Voici un exemple de l'opcode `cpstun`. Il utilise le fichier `cpstun.csd` [examples/cpstun.csd].

### Exemple 99. Exemple de l'opcode `cpstun`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpstun.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
          1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
          1.498, 1.588, 1.682, 1.782, 1.888, 2.000

; Instrument #1.
instr 1
; Set the trigger.
ktrig init 1

; Use Table #1.
kfn init 1

; If the base key (note #60) is C, then 9 notes
; above it (note #60 + 9 = note #69) should be A.
kindex init 69

k1 cpstun ktrig, kindex, kfn

printk2 k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
i1      440.11044
```

## Voir Aussi

*cpstmid*, *cpstuni*, *GEN02*

## Crédits

Exemple écrit par Kevin Conder.

# cpstuni

cpstuni — Retourne des valeurs d'échelle microtonale au taux-i.

## Description

Retourne des valeurs d'échelle microtonale au taux-i.

## Syntaxe

```
icps cpstuni index, ifn
```

## Initialisation

*icps* -- Valeur de retour en cycles par seconde.

*index* -- Un nombre entier servant d'indice dans l'échelle.

*ifn* -- Table de fonction contenant les paramètres (numgrades, interval, basefreq, basekeymidi) ainsi que les rapports de hauteur.

## Exécution

Cet opcode est similaire à *cpstmid*, mais son fonctionnement ne nécessite pas le MIDI.

*cpstuni* travaille au taux-i. Il permet d'obtenir des échelles microtonales personnalisées. Il nécessite le numéro d'une table de fonction qui contient les rapports de hauteur, et quelques autres paramètres stockés dans la table elle-même.

L'argument *index* reçoit un nombre entier indiquant quel degré de l'échelle donnée doit être converti en Hz. La table de fonction *ifn* sera générée par *GEN02*, les quatre premières valeurs stockées dans la table étant des paramètres qui expriment :

- numgrades -- Le nombre de degrés de l'échelle microtonale.
- interval -- L'intervalle de fréquence couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.
- basefreq -- La fréquence de base de l'échelle en cycles par seconde.
- basekey -- L'indice entier dans l'échelle auquel la fréquence de base sera affectée sans changement.

On peut insérer les rapports de hauteur après ces quatre valeurs. Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 Hz affectée au numéro de touche 60, l'instruction f de la partition pour générer la table sera :

```
;          numgrades    basefreq    tuning-ratios (eq.temp) .....  
;          interval      basekey  
f1 0 64 -2 12      2      261    60      1    1.059463 1.12246 1.18920 ..etc...
```

Un autre exemple avec une échelle de 24 degrés et une fréquence de base de 440 affectée au numéro de touche 48, et un intervalle de répétition de 1,5 :

```
;
;          numgrades      basefreq      tuning-ratios .....
;          interval      basekey
f1 0 64 -2      24      1.5      440      48      1      1.01 1.02 1.03 ..etc...
```

## Exemples

Voici un exemple de l'opcode cpstuni. Il utilise le fichier *cpstuni.csd* [examples/cpstuni.csd].

### Exemple 100. Exemple de l'opcode cpstuni.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpstuni.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
          1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
          1.498, 1.588, 1.682, 1.782, 1.888, 2.000

; Instrument #1.
instr 1
; Use Table #1.
ifn = 1

; If the base key (note #60) is C, then 9 notes
; above it (note #60 + 9 = note #69) should be A.
index = 69

i1 cpstuni index, ifn

print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 440.110
```

## Voir Aussi

*cpstmid*, *cpstun*, *GEN02*

## Crédits

Ecrit par Gabriel Maldonado.

Exemple écrit par Kevin Conder.



# cpsxpch

cpsxpch — Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de n'importe quel intervalle.

## Description

Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de n'importe quel intervalle. Le nombre de divisions ne doit pas dépasser 100.

## Syntaxe

icps **cpsxpch** ipch, iequal, irepeat, ibase

## Initialisation

*ipch* -- Nombre en entrée de la forme 8ve.pc, indiquant une octave et quelle note dans l'octave.

*iequal* -- S'il est positif, c'est le nombre d'intervalles égaux de division de l'« octave ». Doit être inférieur ou égal à 100. S'il est négatif, c'est le numéro d'une table de multiplicateurs de fréquence.

*irepeat* -- Nombre indiquant l'intervalle qui est l'« octave ». Le nombre 2 est utilisé pour des divisions de l'octave, 3 pour une douzième, 4 pour deux octaves, ainsi de suite. Ce nombre ne doit pas forcément être un entier, mais il doit être positif.

*ibase* -- La fréquence qui correspond à la hauteur 0.0



### Note

1. Les lignes suivantes sont équivalentes

```
ia = cpspch(8.02)
ib  cps2pch 8.02, 12
ic  cpsxpch 8.02, 12, 2, 1.02197503906
```

2. C'est un opcode, pas une fonction.
3. Des valeurs négatives sont permises pour *ipch*, mais pas pour *irepeat*, ni pour *iequal* ou *ibase*.

## Exemples

Voici un exemple de l'opcode cpsxpch. Il utilise le fichier *cpsxpch.csd* [examples/cpsxpch.csd].

### Exemple 101. Exemple de l'opcode cpsxpch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a normal twelve-tone scale.
ipch = 8.02
iequal = 12
irepeat = 2
ibase = 1.02197503906

icps cpsxpch ipch, iequal, irepeat, ibase

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 293.666
```

Voici un exemple de l'opcode cpsxpch qui utilise une échelle tempérée égale avec 10,5 divisions de l'octave. Il utilise le fichier *cpsxpch\_105et.csd* [examples/cpsxpch\_105et.csd].

### Exemple 102. Exemple de l'opcode cpsxpch qui utilise une échelle tempérée égale avec 10,5 divisions de l'octave.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch_105et.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
```

```

; Use a 10.5ET scale.
ipch = 4.02
iequal = 21
irepeat = 4
ibase = 16.35160062496

icps cpsxpch ipch, iequal, irepeat, ibase

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 4776.824
```

Voici un exemple de l'opcode cpsxpch qui utilise une échelle de Pierce centrée sur le la médian. Il utilise le fichier *cpsxpch\_pierce.csd* [examples/cpsxpch\_pierce.csd].

### Exemple 103. Exemple de l'opcode cpsxpch qui utilise une échelle de Pierce centrée sur le la médian.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch_pierce.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a Pierce scale centered on middle A.
ipch = 2.02
iequal = 12
irepeat = 3
ibase = 261.62561

icps cpsxpch ipch, iequal, irepeat, ibase

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 2827.762
```

## Voir Aussi

*cpspch, cps2pch*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
1997

Nouveau dans la version 3.492 de Csound

# cpuprc

cpuprc — Contrôle l'allocation des ressources cpu par instrument, pour optimiser la sortie en temps réel.

## Description

Contrôle l'allocation des ressources cpu par instrument, pour optimiser la sortie en temps réel.

## Syntaxe

```
cpuprc insnum, ipercent
```

## Initialisation

*insnum* -- numéro de l'instrument

*ipercent* -- pourcentage du temps de traitement cpu à allouer. On peut aussi l'exprimer sous forme fractionnaire.

## Exécution

*cpuprc* fixe le pourcentage du temps de traitement cpu utilisé par un instrument, afin d'éviter un sous-remplissage du tampon dans les exécutions en temps réel. L'utilisateur doit fixer la valeur de *ipercent* pour chaque instrument qui sera activé en temps réel. En supposant que le temps de traitement cpu total soit 100% en théorie, cette valeur de pourcentage ne peut être définie qu'empiriquement, car il y a trop de facteurs qui contribuent à limiter la polyphonie en temps réel sur différents ordinateurs.

Par exemple si *ipercent* est fixé à 5% pour l'instrument 1, le nombre maximum de voix que l'on pourra allouer en temps réel est 20 ( $5\% * 20 = 100\%$ ). Si l'on essaye de jouer une note supplémentaire alors que les 20 notes précédentes sont toujours jouées, Csound empêche l'allocation de cette note et affiche le message d'avertissement suivant :

impossible d'allouer la dernière note car il n'y a plus de temps cpu disponible

Afin d'éviter les sous-remplissages de tampon audio, il est suggéré de fixer le nombre maximum de voix à une valeur légèrement inférieure à la puissance de traitement réelle de l'ordinateur. Parfois, un instrument peut avoir besoin de plus de temps de traitement que la normale. Si, par exemple, l'instrument contient un oscillateur qui lit une table trop grande pour la mémoire cache, il sera plus lent qu'en temps normal. De plus, chaque programme s'exécutant concurremment dans l'environnement multitâche, peut consommer de la puissance processeur à divers degrés.

Au début, tous les instruments reçoivent une valeur par défaut de *ipercent* égale à 0.0% (c'est-à-dire un temps de traitement nul équivalent à une vitesse de processeur infinie). Ce réglage convient très bien pour les sessions en temps différé.

Toutes les instances de *cpuprc* doivent être définies dans la section d'en-tête et non dans le corps de l'instrument.

## Exemples

Voici un exemple de l'opcode `cpuprc`. Il utilise le fichier `cpuprc.csd` [examples/cpuprc.csd].

### Exemple 104. Exemple de l'opcode `cpuprc`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpuprc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Limit Instrument #1 to 5% of the CPU processing time.
cpuprc 1, 5

; Instrument #1
instr 1
  al oscil 10000, 440, 1
  out al
endin

</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*maxalloc, prealloc*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# cross2

cross2 — Synthèse croisée au moyen de TFR.

## Description

C'est une implémentation de synthèse croisée au moyen de TFR.

## Syntaxe

```
ares cross2 ain1, ain2, isize, ioverlap, iwin, kbias
```

## Initialisation

*isize* -- Taille de la TFR à effectuer. Plus la taille est grande, meilleure est la réponse en fréquence mais avec une réponse temporelle imprécise.

*ioverlap* -- Facteur de chevauchement des TFR, doit être une puissance de deux. Les meilleurs réglages sont 2 et 4. Un grand chevauchement prend un long temps de calcul.

*iwin* -- Table de fonction contenant la fenêtre à utiliser dans l'analyse. On peut créer cette fenêtre au moyen de la routine *GEN20*.

## Exécution

*ain1* -- Le son d'excitation. Doit contenir des fréquence élevées pour de meilleurs résultats.

*ain2* -- Le son modulant. Doit avoir une réponse en fréquence changeante (comme la parole) pour de meilleurs résultats.

*kbias* -- la proportion de synthèse croisée. 1 est la normale, 0 signifie pas de synthèse croisée.

## Exemples

Voici un exemple de l'opcode cross2. Il utilise les fichiers *cross2.csd* [examples/cross2.csd] et *beats.wav* [examples/beats.wav].

### Exemple 105. Exemple de l'opcode cross2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cross2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - Play an audio file.
instr 1
; Use the "fox.wav" audio file.
aout soundin "fox.wav"
out aout
endin

; Instrument #2 - Cross-synthesize!
instr 2
; Use the "ahh" sound stored in Table #1.
icps = p4
ain1 oscil 30000, p4, 1
; Use the "beats.wav" audio file.
ain2 soundin "fox.wav"

isize = 4096
ioverlap = 2
iwin = 2
kbias init 1

aout cross2 ain1, ain2, isize, ioverlap, iwin, kbias

out aout
endin

</CsInstruments>
<CsScore>

; Table #1: An audio file.
f 1 0 128 1 "eee.aiff" 0 4 0
; Table #2: A windowing function.
f 2 0 2048 20 2

; Play Instrument #1 for 3 seconds.
i 1 0 3
; Play Instrument #2 with various frequencies for "carrier"
i 2 3 3 50
i 2 6 3 100
i 2 9 3 250
i 2 12 3 20
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1997

Exemple écrit par Kevin Conder.



# crossfm

crossfm — Deux oscillateurs se modulant mutuellement en fréquence et/ou en phase.

## Description

Deux oscillateurs se modulant mutuellement en fréquence et/ou en phase.

## Syntaxe

```
a1, a2 crossfm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]

a1, a2 crossfmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]

a1, a2 crossspm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]

a1, a2 crossspmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]

a1, a2 crossfmpm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]

a1, a2 crossfmpmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
```

## Initialisation

*ifn1* -- numéro de la table de fonction pour l'oscillateur 1. Nécessite un point de garde.

*ifn2* -- numéro de la table de fonction pour l'oscillateur 2. Nécessite un point de garde.

*iphs1* (facultatif, 0 par défaut=0) -- phase initiale de la forme d'onde de la table *ifn1*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation sera ignorée.

*iphs2* (facultatif, 0 par défaut=0) -- phase initiale de la forme d'onde de la table *ifn2*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation sera ignorée.

## Exécution

*xfrq1* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps* donne la fréquence de l'oscillateur 1.

*xfrq2* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps* donne la fréquence de l'oscillateur 2.

*xndx1* -- l'indice de modulation de l'oscillateur 2 par l'oscillateur 1.

*xndx2* -- l'indice de modulation de l'oscillateur 1 par l'oscillateur 2.

*kcps* -- un dénominateur commun, en cycles par seconde, pour les fréquences des deux oscillateurs.

*crossfm* implémente un algorithme de modulation de fréquence croisée. La sortie de taux audio de l'oscillateur 1 module l'entrée en fréquence de l'oscillateur 2 tandis que la sortie audio de l'oscillateur 2 module l'entrée en fréquence de l'oscillateur 1. Cette double structure de rétroaction produit des sonorités riches avec parfois un comportement chaotique. *crossfmi* fonctionne comme *crossfm* mais en utilisant l'interpolation linéaire pour la lecture des tables.

*crosspm* et *crosspmi* implémentent la modulation de phase croisée entre deux oscillateurs.

*crossfmpm* et *crossfmpmi* implémentent une modulation de fréquence/phase croisée entre deux oscillateurs. L'oscillateur 1 est modulé en fréquence par l'oscillateur 2 tandis que l'oscillateur 2 est modulé en phase par l'oscillateur 1.

On peut lire mon *article* [<http://www.csounds.com/journal/issue12/crossfm.html>] dans le Csound Journal pour plus d'information.



## Avertissement

Ces opcodes peuvent produire des spectres très riches, particulièrement avec des indices de modulation importants et, dans certains cas des fréquences de repliement peuvent apparaître si le taux d'échantillonnage n'est pas suffisamment élevé. De plus, la sortie audio peut varier en fonction du taux d'échantillonnage à cause de la non-linéarité de l'algorithme. Dans Csound, deux autres opcodes présentent cette caractéristique : *planet* et *chuap*.

## Exemples

Voici un exemple de l'opcode *crossfm*. Il utilise le fichier *crossfm.csd* [examples/oscil.csd].

### Exemple 106. Exemple de l'opcode *crossfm*.

```
<CsoundSynthesizer>
<CsOptions>
  -d -o dac
</CsOptions>
<CsInstruments>
sr          =          96000
ksmps       =          10
nchnls      =          2
odbfs       =          1

FLpanel "crossfmForm", 600, 400, 0, 0
  gkfrq1, ihfrq1 FLcount "Freq #1", 0, 20000, 0.001, 1, 1, 200, 30, 20, 50, -1
  gkfrq2, ihfrq2 FLcount "Freq #2", 0, 20000, 0.001, 1, 1, 200, 30, 20, 130, -1
  gkndx1, gkndx2, ihndx1, ihndx2 FLjoy "Indexes", 0, 10, 0, 10, 0, 0, -1, -1, 200, 200, 300, 50

  FLsetVal_i 164.5, ihfrq1
  FLsetVal_i 263.712, ihfrq2
  FLsetVal_i 1.5, ihndx1
  FLsetVal_i 3, ihndx2
FLpanelEnd
FLrun

maxalloc 1, 2

instr 1
  kamp      linen      0.5, 0.01, p3, 0.5
  a1,a2     crossfm     gkfrq1, gkfrq2, gkndx1, gkndx2, 1, 1, 1
  outs      a1*kamp, a2*kamp
endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1 0
i1 0 60
e
</CsScore>
</CsoundSynthesizer>
```

Dans cet exemple, on utilise une interface graphique FLTK pour contrôler en temps réel la fréquence

des oscillateurs au moyen de deux contrôles *Flcount* et les indices de la modulation croisée avec un contrôle *FLjoy*. Il utilise un taux d'échantillonnage de 96000Hz.

## Crédits

Auteur : François Pinot  
2005-2009

Nouveau dans la version 5.12

# crunch

crunch — Modèle semi-physique d'un son de craquement.

## Description

*crunch* est un modèle semi-physique d'un son de craquement. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntax

```
ares crunch iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 7.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,998 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,99806 ce qui signifie que la valeur par défaut de *idamp* est 0,03. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode crunch. Il utilise le fichier *crunch.csd* [examples/crunch.csd].

### Exemple 107. Exemple de l'opcode crunch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;;RT audio I/O
```

```

; For Non-realtime ouput leave only the line below:
; -o crunch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra -----

    sr =          44100
    kr =          4410
    ksmpr =       10
    nchnls =      1

instr 01                ;an example of a crunch
a1      crunch p4, 0.01
        out a1
        endin

</CsInstruments>
<CsScore>

;score -----

i1 0 1 26000
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*cabasa, sandpaper, sekere, stix*

## Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

## ctrl14

ctrl14 — Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest ctrl14 ichan, ictlno1, ictlno2, imin, imax [, ifn]
```

```
kdest ctrl14 ichan, ictlno1, ictlno2, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ichan* -- numéro de canal MIDI (1-16)

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*imin* -- la valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- la valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- la valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- la valeur décimale maximale de sortie définie par l'utilisateur

*ctrl14* (contrôle MIDI sur 14 bit au taux-i et au taux-k) permet un signal MIDI sur 14 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table. Il nécessite deux contrôleurs MIDI en entrée.

*ctrl14* est différent de *midic14* parce que il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur. Le canal MIDI est le même pour tous les contrôleurs utilisés dans un opcode *ctrl14*.

## Voir aussi

*ctrl7, ctrl21, initc7, initc14, initc21, midic7, midic14, midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# ctrl21

**ctrl21** — Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest ctrl21 ichan, ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
```

```
kdest ctrl21 ichan, ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ichan* -- numéro de canal MIDI (1-16)

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids moyen (0-127)

*ictlno3* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*imin* -- la valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- la valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- la valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- la valeur décimale maximale de sortie définie par l'utilisateur

*ctrl21* (contrôle MIDI sur 21 bit au taux-i et au taux-k) permet un signal MIDI sur 21 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser une indexation de table facultative. Il nécessite trois contrôleurs MIDI en entrée.

*ctrl21* est différent de *midic21* parce qu'il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur. Le canal MIDI est le même pour tous les contrôleurs utilisés dans un opcode *ctrl21*.



## Voir aussi

*ctrl7, ctrl14, initc7, initc14, initc21, midic7, midic14, midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# ctrl7

**ctrl7** — Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

`idest ctrl7 ichan, ictlno, imin, imax [, ifn]`

`kdest ctrl7 ichan, ictlno, kmin, kmax [, ifn]`

`adest ctrl7 ichan, ictlno, kmin, kmax [, ifn] [, icutoff]`

## Initialisation

*idest* -- signal de sortie

*ichan* -- canal MIDI (1-16)

*ictlno* -- numéro du contrôleur MIDI (0-127)

*imin* -- valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imin* et *imax*.

*icutoff* (facultatif) -- fréquence de coupure du filtre passe-bas pour lisser la sortie au taux-a.

## Exécution

*kdest*, *adest* -- signal de sortie

*kmin* -- valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ctrl7* (contrôle MIDI sur 7 bit au taux-i et au taux-k) permet un signal MIDI sur 7 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Il permet également en option une indexation de table sans interpolation. Les valeurs minimale et maximale peuvent varier au taux-k.

*ctrl7* diffère de *midic7* parce que il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a aussi besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur.

La version de *ctrl7* au taux-a fournit en sortie une variable de taux-a, qui est passée par un filtre passe-bas (lissée). Il y a un paramètre facultatif *icutoff*, pour établir la fréquence de coupure du filtre passe-bas.

Sa valeur par défaut est 5.

## Voir aussi

*ctrl14, ctrl21, initc7, initc14, initc21, midic7, midic14, midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

La version de *ctrl7* au taux-a a été ajoutée dans la version 5.06

# ctrlinit

ctrlinit — Initialise les valeurs pour un groupe de contrôleurs MIDI.

## Description

Initialise les valeurs pour un groupe de contrôleurs MIDI.

## Syntaxe

```
ctrlinit ichnl, ictlno1, ival1 [, ictlno2] [, ival2] [, ictlno3] \  
[, ival3] [...ival32]
```

## Initialisation

*ichnl* -- numéro de canal MIDI (1-16)

*ictlno1*, *ictlno1*, etc. -- numéros de contrôleurs MIDI (0-127)

*ival1*, *ival2*, etc. -- valeur initiale pour le contrôleur MIDI de numéro correspondant

## Exécution

Initialise les valeurs pour un groupe de contrôleurs MIDI.

## Voir aussi

*massign*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT, Cambridge, Mass.

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# cuserrnd

cuserrnd — Générateur de nombres aléatoires de distribution continue définie par l'utilisateur.

## Description

Générateur de nombres aléatoires de distribution continue définie par l'utilisateur.

## Syntaxe

aout **cuserrnd** kmin, kmax, ktableNum

iout **cuserrnd** imin, imax, itableNum

kout **cuserrnd** kmin, kmax, ktableNum

## Initialisation

*imin* -- limite inférieure de l'intervalle

*imax* -- limite supérieure de l'intervalle

*itableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

## Exécution

*ktableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

*cuserrnd* (Continuous USER-defined-distribution RaNDom generator) génère des nombres aléatoires selon une distribution aléatoire continue créée par l'utilisateur. Dans ce cas la forme de l'histogramme de la distribution peut être dessinée ou générée par n'importe quelle routine GEN. La table contenant la forme d'un tel histogramme doit être transformée ensuite en une fonction de distribution au moyen de GEN40 (voir GEN40 pour plus de détails). Cette fonction doit ensuite être allouée à l'argument *XtableNum* de *cuserrnd*. L'intervalle de sortie sera ensuite mis à l'échelle selon les arguments *Xmin* et *Xmax*. *cuserrnd* faisant une interpolation linéaire entre les éléments de la table, il n'est pas recommandé pour les distributions discrètes (GEN41 et GEN42).

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Voir Aussi

*dusernd, urd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16

# dam

dam — Un compresseur/expander dynamique.

## Description

Cet opcode modifie dynamiquement une valeur de gain appliquée à l'entrée *ain* en comparant son niveau de puissance à un seuil de niveau donné. Le signal est compressé ou élargi de différents facteurs selon qu'il est au-dessus ou en-dessous du seuil.

## Syntaxe

```
ares dam asig, kthreshold, icomp1, icomp2, irtime, iftime
```

## Initialisation

*icomp1* -- rapport de compression pour la zone supérieure.

*icomp2* -- rapport de compression pour la zone inférieure

*irtime* -- montée du gain en secondes. Durée au-delà de laquelle le facteur de gain peut augmenter d'une unité.

*iftime* -- chute du gain en secondes. Durée au-delà de laquelle le facteur de gain peut diminuer d'une unité. of one unit.

## Exécution

*asig* -- signal d'entrée à modifier

*kthreshold* -- niveau du signal d'entrée qui agit comme seuil. Il peut changer au taux-k (par exemple pour le ducking)

Note sur les taux de compression : un taux de compression de un laisse le son inchangé. Avec un rapport inférieur à un, le signal sera compressé (réduction de son volume) tandis qu'avec un rapport supérieur à un, le signal sera élargi (augmentation de son volume).

## Exemples

Les résultats de l'opcode *dam* pouvant être subtils, je recommande de les regarder dans un éditeur de sons graphique comme *audacity*. *audacity* existe pour Linux, Windows, et MacOS et on peut le télécharger à <http://audacity.sourceforge.net> [<http://audacity.sourceforge.net/>].

Voici un exemple de l'opcode *dam*. Il utilise les fichiers *dam.csd* [examples/dam.csd] et *beats.wav* [examples/beats.wav].

### Exemple 108. Un exemple de l'opcode *dam* compressant un signal audio.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
;-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o dam.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1, uncompressed signal.
instr 1
; Use the "beats.wav" audio file.
asig soundin "beats.wav"

out asig
endin

; Instrument #2, compressed signal.
instr 2
; Use the "beats.wav" audio file.
asig soundin "beats.wav"

; Compress the audio signal.
kthreshold = 25000
icompl = 0.5
icomp2 = 0.763
irtime = 0.1
iftime = 0.1
al dam asig, kthreshold, icompl, icomp2, irtime, iftime

out al
endin

; Instrument #3, compressed signal.
instr 3
; Use the "beats.wav" audio file.
asig soundin "beats.wav"

; Compress the audio signal.
kthreshold line 25000, p3, 4410000
icompl = 0.5
icomp2 = 0.763
irtime = 0.1
iftime = 0.1
al dam asig, kthreshold, icompl, icomp2, irtime, iftime

out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 2 2
; Play Instrument #3 for 2 seconds.
i 3 4 2
e

</CsScore>
</CsoundSynthesizer>

```

Cet exemple compresse le fichier audio « beats.wav ». On y entend un schéma de batterie répété deux fois. La deuxième fois, le son est moins fort (compressé) que la première fois.

Voici un autre exemple de l'opcode dam. Il utilise les fichiers *dam\_expanded.csd* [examples/



dam\_expanded.csd] et *mary.wav* [examples/mary.wav].

### Exemple 109. Un exemple de l'opcode dam élargissant un signal audio.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o dam_expanded.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1, normal audio signal.
instr 1
; Use the "mary.wav" audio file.
asig soundin "mary.wav"

out asig
endin

; Instrument #2, expanded audio signal.
instr 2
; Use the "mary.wav" audio file.
asig soundin "mary.wav"

; Expand the audio signal.
kthreshold init 7500
icompl = 2.25
icomp2 = 2.25
irtime = 0.1
ifetime = 0.6
al dam asig, kthreshold, icompl, icomp2, irtime, iftime

out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1.
i 1 0.0 3.5
; Play Instrument #2.
i 2 3.5 3.5
e

</CsScore>
</CsoundSynthesizer>
```

Cet exemple élargit le fichier audio « mary.wav ». On y entend une mélodie répétée deux fois. La deuxième fois, le son est plus fort (élargi) que la première fois.

## Crédits

Auteur : Marc Resibois  
Belgique  
1997

Nouveau dans la version 3.47

Exemples écrits par Kevin Conder.

# date

date — Retourne le nombre de secondes écoulées depuis le 1er janvier 1970.

## Description

Retourne le nombre de secondes écoulées depuis le 1er janvier 1970, en lisant l'horloge du système d'exploitation.

## Syntaxe

```
ir date
```

## Initialisation

*ir --* valeur en secondes à l'initialisation de la note de l'horloge système depuis le début de l'epoch.

## Exemples

Voici un exemple de l'opcode date. Il utilise le fichier *date.csd* [examples/date.csd].

### Exemple 110. Exemple de l'opcode date.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o date.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
instr 1
  ii date
  print ii
  Sa dates ii
  prints Sa
  Ss dates -1
  prints Ss
  St dates 1
  prints St
endin
</CsInstruments>
<CsScore>
i 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  ii = 1165665152.000
Sat Dec  9 11:52:32 2006
Sat Dec  9 11:51:46 2006
Thu Jan  1 01:00:01 1970
```

## Voir Aussi

*dates*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Décembre 2006

Nouveau dans la version 5.05 de Csound

# dates

dates — Retourne sous forme de chaîne de caractères la date et l'heure spécifiées.

## Description

Retourne sous forme de chaîne de caractères la date et l'heure spécifiées.

## Syntaxe

`Sir dates [ itime]`

## Initialisation

*itime* -- le temps écoulé en secondes depuis le début de l'epoch. S'il est omis ou s'il est négatif, le temps courant est utilisé.

*Sir* -- la date et l'heure sous forme de chaîne de caractères.

## Exemples

Voici un exemple de l'opcode dates. Il utilise le fichier *date.csd* [examples/date.csd].

### Exemple 111. Exemple de l'opcode dates.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o date.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
instr 1
  ii date
  print ii
  Sa dates ii
  prints Sa
  Ss dates -1
  prints Ss
  St dates 1
  prints St
endin

</CsInstruments>
<CsScore>
i 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  ii = 1165665152.000
Sat Dec  9 11:52:32 2006
Sat Dec  9 11:51:46 2006
```

Thu Jan 1 01:00:01 1970

## Voir Aussi

*date*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Décembre 2006

Nouveau dans la version 5.05 de Csound

# db

db — Retourne l'amplitude équivalente pour une valeur donnée en décibels.

## Description

Retourne l'amplitude équivalente pour une valeur donnée en décibels. Cet opcode est le même que *ampdb*.

## Syntaxe

`db(x)`

Cette fonction fonctionne aux taux-i, -k et -a.

## Initialisation

*x* -- une valeur exprimée en décibels.

## Exécution

Retourne l'amplitude équivalente pour une valeur donnée en décibels.

## Exemples

Voici un exemple de l'opcode db. Il utilise le fichier *db.csd* [examples/db.csd].

### Exemple 112. Example of the db opcode.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o db.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Calculate the amplitude of 40 decibels.
idecibels = 40
iamp = db(idecibels)

print iamp
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  iamp = 100.000
```

## Voir Aussi

*ampdb, cent, octave, semitone*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.16



# dbamp

dbamp — Retourne l'équivalent en décibel de l'amplitude  $x$ .

## Description

Retourne l'équivalent en décibel de l'amplitude  $x$ .

## Syntaxe

`dbamp(x)` (arguments de taux-i ou -k seulement)

## Exemples

Voici un exemple de l'opcode dbamp. Il utilise le fichier *dbamp.csd* [examples/dbamp.csd].

### Exemple 113. Exemple de l'opcode dbamp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dbamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 30000
  idb = dbamp(iamp)

  print idb
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  idb = 89.542
```

## Voir Aussi

*ampdb, ampdbfs, dbfsamp*

## Crédits

Exemple écrit par Kevin Conder.

# dbfsamp

dbfsamp — Retourne l'équivalent en décibel de l'amplitude  $x$ , relative à l'amplitude maximale.

## Description

Retourne l'équivalent en décibel de l'amplitude  $x$ , relative à l'amplitude maximale. L'amplitude maximale est supposée être codée en 16 bit. Nouveau dans la version 4.10.

## Syntaxe

**dbfsamp**( $x$ ) (arguments de `taux-i` ou `-k` seulement)

## Exemples

Voici un exemple de l'opcode `dbfsamp`. Il utilise le fichier `dbfsamp.csd` [exemples/dbfsamp.csd].

### Exemple 114. Exemple de l'opcode `dbfsamp`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o dbfsamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 30000
  idb = dbfsamp(iamp)

  print idb
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1: idb = -0.767
```

## Voir Aussi

*ampdb, ampdbfs, dbamp*

## Crédits

Exemple écrit par Kevin Conder.

# dcblock

dcblock — A DC blocking filter.

## Description

Implements the DC blocking filter

$$Y[i] = X[i] - X[i-1] + (igain * Y[i-1])$$

Based on work by Perry Cook.

## Syntax

ares **dcblock** ain [ , igain]

## Initialization

*igain* -- the gain of the filter, which defaults to 0.99

## Performance

*ain* -- audio signal input



### Note

The new *dcblock2* opcode is an improved method of removing DC from an audio signal.

## Examples

Here is an example of the *dcblock* opcode. It uses the file *dcblock.csd* [examples/dcblock.csd], and *beats.wav* [examples/beats.wav].

### Exemple 115. Example of the *dcblock* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dcblock.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 -- normal audio signal.
instr 1
  asig soundin "beats.wav"
  out asig
endin

; Instrument #2 -- dcblock-ed audio signal.
instr 2
  asig soundin "beats.wav"

  igain = 0.75
  al dcblock asig, igain

  out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## See also

*dcblock2*

## Credits

Author: John ffitch  
 University of Bath, Codemist Ltd.  
 Bath, UK

Example written by Kevin Conder.

New in Csound version 3.49

February 2003: Thanks to a note from Anders Andersson, corrected the formula.

# dcblock2

dcblock2 — Un filtre bloqueur de composante continue.

## Description

Implémente un filtre bloqueur de composante continue avec une atténuation améliorée de la composante continue.

## Syntaxe

```
ares dcblock2 ain [, iorder] [, iskip]
```

## Initialisation

*iorder* -- ordre du filtre, au minimum 4ème ordre, vaut par défaut 128.

*iskip* -- s'il vaut 1, l'initialisation est ignorée (0 par défaut).

## Exécution

*ares* -- signal audio filtré

*ain* -- signal audio en entrée



### Note

Avec l'utilisation d'une valeur inférieure à *ksmps* pour *iorder*, la réduction du décalage dû à la composante continue ne sera pas efficace.

## Voir Aussi

*dcblock*

## Crédits

Par Victor Lazzarini

Nouveau dans la version 5.09 de Csound

# dconv

dconv — Un opcode de convolution directe.

## Description

Un opcode de convolution directe.

## Syntaxe

```
ares dconv asig, isize, ifn
```

## Initialisation

*isize* -- la taille du tampon de convolution à utiliser. Si la taille du tampon est inférieure à celle de *ifn*, alors seules les premières *isize* valeurs de la table seront utilisées.

*ifn* -- numéro de la table d'une fonction stockée contenant la réponse impulsionnelle pour la convolution.

## Exécution

Plutôt que d'utiliser la méthode d'analyse/resynthèse de l'opcode *convolve*, *dconv* utilise la convolution directe pour créer le résultat. Avec de petites tables, il peut faire cela avec une certaine efficacité, alors que des tables plus grandes nécessitent bien plus de temps de calcul. *dconv* effectue (*isize* \* *ksmps*) multiplications à chaque cycle-k. C'est pourquoi les effets de réverbération et de délai sont mieux réalisés par d'autre opcodes (à moins que les durées soient courtes).

*dconv* a été conçu pour travailler avec des tables variant dans le temps pour faciliter de nouvelles possibilités de filtrage en temps réel.

## Exemples

Voici un exemple de l'opcode *dconv*. Il utilise le fichier *dconv.csd* [examples/dconv.csd].

### Exemple 116. Exemple de l'opcode dconv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dconv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```



```

#define RANDI(A) #kout randi 1, kfq, $A*.001+iseed, 1
        tablew kout, $A, itable#

instr 1
itable init 1
iseed init .6
isize init ftlen(itable)
kfq line 1, p3, 10

$RANDI(0)
$RANDI(1)
$RANDI(2)
$RANDI(3)
$RANDI(4)
$RANDI(5)
$RANDI(6)
$RANDI(7)
$RANDI(8)
$RANDI(9)
$RANDI(10)
$RANDI(11)
$RANDI(12)
$RANDI(13)
$RANDI(14)
$RANDI(15)

asig rand 10000, .5, 1
asig butlp asig, 5000
asig dconv asig, isize, itable

        out asig *.5
endin

</CsInstruments>
<CsScore>

f1 0 16 10 1
i1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*pconvolve, convolve, ftconv*

## Crédits

Auteur : William « Pete » Moss  
2001

Nouveau dans la version 4.12

# delay

delay — Retarde un signal d'entrée d'une certaine durée.

## Description

Un signal peut être lu ou écrit dans une ligne à retard, ou il peut être retardé automatiquement d'une certaine durée.

## Syntaxe

```
ares delay asig, idlt [, iskip]
```

## Initialisation

*idlt* -- délai demandé en secondes. Il peut être aussi grand que la mémoire disponible le permet. L'espace requis pour *n* secondes de délai est de  $4n * sr$  octets. Il est alloué lorsque l'instrument est initialisé pour la première fois, et retourne dans le pool à la fin d'une section de partition.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace des données de la boucle de retard (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio

*delay* est un composé de *delayr* et de *delayw*, écrivant et lisant à la fois dans son propre espace de stockage. Il peut ainsi accomplir un décalage temporel du signal, bien que la rétroaction variable ne soit pas possible. Il n'y a pas de durée de délai minimale.

## Exemples

Voici un exemple de l'opcode *delay*. Il utilise le fichier *delay.csd* [examples/delay.csd].

### Exemple 117. Exemple de l'opcode *delay*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o delay.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
```

```
; Instrument #1 -- Delayed beeps.
instr 1
; Make a basic sound.
abep vco 20000, 440, 1

; Delay the beep by .1 seconds.
idlt = 0.1
adel delay abep, idlt

; Send the beep to the left speaker and
; the delayed beep to the right speaker.
outs abep, adel
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Keep the score running for 2 seconds.
f 0 2

; Play Instrument #1.
i 1 0.0 0.2
i 1 0.5 0.2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*delayl, delayr, delayw*

## Crédits

Exemple écrit par Kevin Conder.

# delay1

delay1 — Retarde un signal d'entrée d'un échantillon.

## Description

Retarde un signal d'entrée d'un échantillon.

## Syntaxe

```
ares delay1 asig [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace des données de la boucle de retard (voir *reson*). La valeur par défaut est 0.

## Exécution

*delay1* est une forme spéciale de délai qui sert à retarder le signal audio *asig* d'un seul échantillon. Il est ainsi fonctionnellement équivalent à l'opcode *delay* mais il est plus efficace à la fois en temps et en espace. Cette unité est particulièrement utile dans la fabrication de filtres non récurrents généralisés.

## Voir Aussi

*delay*, *delayr*, *delayw*

# delayk

delayk — Retarde un signal d'entrée d'une certaine durée.

## Description

Opcodes de retard de taux-k.

## Syntaxe

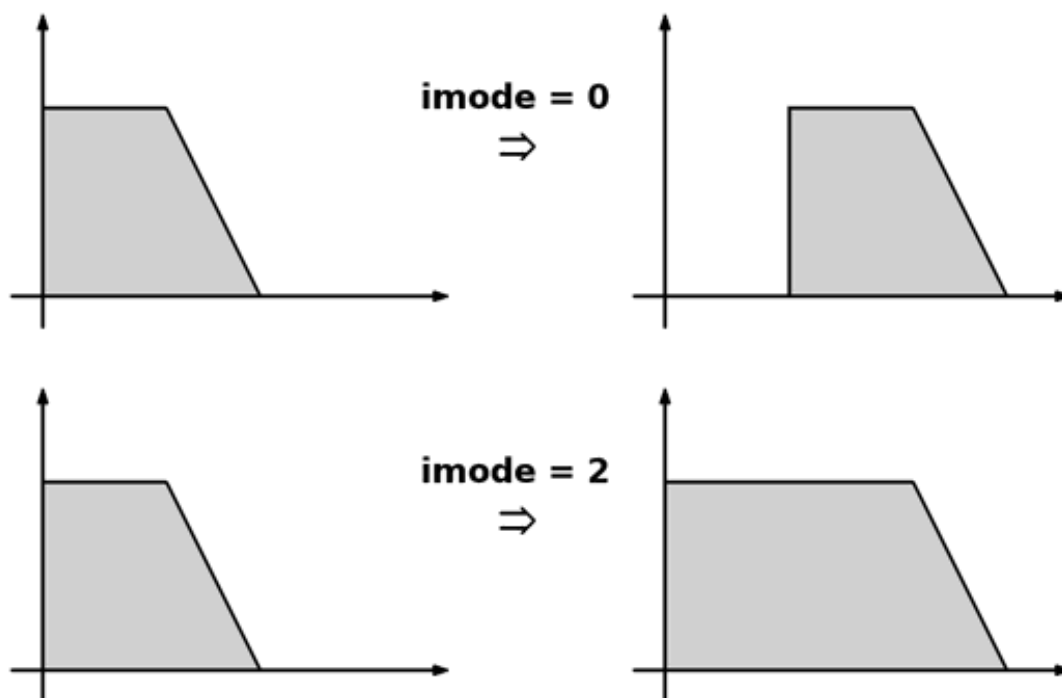
```
kr delayk    ksig, idel[, imode]
```

```
kr vdel_k    ksig, kdel, imdel[, imode]
```

## Initialisation

*idel* -- délai (en secondes) pour *delayk*. Il est arrondi au multiple entier le plus proche d'un cycle-k (c-à-d  $1/kr$ ).

*imode* -- somme de 1 pour ignorer l'initialisation (par exemple pour les notes liées) et de 2 pour maintenir la première valeur d'entrée durant le délai initial au lieu de sortir des zéros. Cela est utile principalement pour retarder des enveloppes qui ne commencent pas à zéro.



*imdel* -- délai maximum pour *vdel\_k*, en secondes.

## Exécution

*kr* -- le signal de sortie. Note : ces opcodes n'interpolent pas leur sortie.

*ksig* -- le signal d'entrée.

*kdel* -- délai (en secondes) pour *vdel\_k*. Il est arrondi au multiple entier le plus proche d'un cycle-k (c-à-d  $1/kr$ ).

## Crédits

Istvan Varga.

# delayr

delayr — Lit depuis une ligne à retard numérique établie automatiquement.

## Description

Lit depuis une ligne à retard numérique établie automatiquement.

## Syntaxe

```
ares delayr idlt [ , iskip]
```

## Initialisation

*idlt* -- délai demandé en secondes. Il peut être aussi grand que la mémoire disponible le permet. L'espace requis pour n secondes de délai est de  $4n * sr$  octets. Il est alloué lorsque l'instrument est initialisé pour la première fois, et retourne dans le pool à la fin d'une section de partition.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace des données de la boucle de retard (voir *reson*). La valeur par défaut est 0.

## Exécution

*delayr* lit depuis une ligne à retard numérique établie automatiquement, dans laquelle le signal restitué est resté pendant *idlt* secondes. Cette unité doit être appariée avec une unité *delayw* qu'elle précède. Il peut y avoir d'autres opcodes de Csound entre les deux.

## Exmples

Voir l'exemple de *delayw*.

## Voir Aussi

*delay*, *delayl*, *delayw*

# delayw

delayw — Écrit le signal audio dans une ligne à retard numérique.

## Description

Écrit le signal audio dans une ligne à retard numérique.

## Syntaxe

```
delayw asig
```

## Exécution

*delayw* écrit *asig* dans l'espace du délai établi par l'unité *delayr* précédente. Cette paire d'unités permet la formation de boucles de rétroaction modifiées, etc. Cependant, il y a une limite inférieure à *idlt*, qui doit valoir au moins une période de contrôle (ou  $1/kr$ ).

## Exemples

Voici un exemple de l'opcode *delayw*. Il utilise le fichier *delayw.csd* [examples/delayw.csd].

### Exemple 118. Exemple de l'opcode *delayw*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o delayw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1 -- Delayed beeps.
instr 1
; Make a basic sound.
abep vco 20000, 440, 1

; Set up a delay line.
idlt = 0.1
adel delayr idlt

; Write the beep to the delay line.
delayw abep

; Send the beep to the left speaker and
; the delayed beep to the right speaker.
outs abep, adel
endin
```



```
</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Keep the score running for 2 seconds.
f 0 2

; Play Instrument #1.
i 1 0.0 0.2
i 1 0.5 0.2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*delay, delayl, delayr*

## Crédits

Exemple écrit par Kevin Conder.

# deltap

deltap — Lit une ligne à retard avec des délais variables.

## Description

Lit une ligne à retard avec des délais variables.

## Syntaxe

```
ares deltap kdlt
```

## Exécution

*kdlt* -- spécifie le délai de lecture en secondes. Chaque valeur est comprise entre 1 période de contrôle et le délai total de la paire lecture/écriture ; cependant, comme il n'y a pas de vérification interne du respect de cet intervalle, l'utilisateur est entièrement responsable. Chaque argument peut être une constante, une variable ou un signal variant dans le temps.

*deltap* extrait le son en lisant directement les échantillons stockés.

Cet opcode peut lire directement dans une paire *delayr/delayw* en extrayant les données audio retardées des *idlt* secondes de son stocké. Il peut y avoir n'importe quel nombre d'unités *deltap* et/ou *deltapi* entre une paire lecture/écriture. Chacune reçoit un extrait audio sans changement de l'amplitude originale.

Ces opcodes peuvent fournir de multiples lectures de délai pour des lignes à retard arbitraires et des réseaux à rétroaction. Ils peuvent délivrer des délais constants ou variables, et sont utiles pour construire des effets de chorus, harmonizer et Doppler. Les délais constants (et certains de ceux qui varient lentement) ne nécessitent pas d'interpolation ; *deltap* leur convient très bien. Les délais variant à moyenne vitesse ou rapidement nécessiteront cependant les services supplémentaires de *deltapi*.

Les paires *delayr/delayw* peuvent être entrelacées. Pour associer une unité de lecture de délai à une unité *delayr*, elle doit non seulement être située entre ce *delayr* et le *delayw* approprié, mais aussi précéder tous les *delayr* suivants. Voir l'exemple 2. (Cette possibilité fut ajoutée dans la version 3.57 de Csound par Jens Groh et John ffitich).

*N.B.* Les délais de taux-k ne sont pas interpolés en interne, mais déroulent plutôt des décalages temporels d'échantillons audios ; c'est adéquat pour des délais changeant lentement. Cependant, pour les changements à vitesse moyenne ou rapides, il faut fournir en entrée des valeurs de délai avec une plus grande résolution de taux audio.

## Exemples

### Exemple 119. Exemple n°1 de deltap

```
asource buzz      1, 440, 20, 1
atime linseg     1, p3/2,.01, p3/2,1 ; trace a distance in secs
ampfac =          1/atime/atime       ; and calc an amp factor
adump delayr     1                   ; set maximum distance
amove deltapi    atime                ; move sound source past
delayw          asource              ; the listener
```

```
out      amove * amfac
```

## Exemple 120. Exemple n°2 de deltap

```

ainput1 = .....
ainput2 = .....
kdlyt1  = .....
kdlyt2  = .....

;Read delayed signal, first delayr instance:
adump   delayr 4.0
adly1   deltap kdlyt1      ; associated with first delayr instance

;Read delayed signal, second delayr instance:
adump   delayr 4.0
adly2   deltap kdlyt2      ; associated with second delayr instance

;Do some cross-coupled manipulation:
afdbk1  =      0.7 * adly1 + 0.7 * adly2 + ainput1
afdbk2  =     -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
        delayw afdbk1

;Feed back signal, associated with second delayr instance:
        delayw afdbk2
        outs   adly1, adly2

```

## Voir Aussi

*deltap3, deltapi, deltapi*

# deltap3

deltap — Lit une ligne à retard avec des délais variables et interpolation cubique.

## Description

Lit une ligne à retard avec des délais variables et interpolation cubique.

## Syntaxe

```
ares deltap3 xdl t
```

## Exécution

*xdl t* -- spécifie le délai de lecture en secondes. Chaque valeur est comprise entre 1 période de contrôle et le délai total de la paire lecture/écriture ; cependant, comme il n'y a pas de vérification interne du respect de cet intervalle, l'utilisateur est entièrement responsable. Chaque argument peut être une constante, une variable ou un signal variant dans le temps ; l'argument *xdl t* de *deltap3* implique qu'une valeur de délai variant au taux audio est autorisée ici.

*deltap3* est expérimental, et utilise l'interpolation cubique. (Nouveau dans la version 3.50 de Csound).

Cet opcode peut lire directement dans une paire *delayr/delayw* en extrayant les données audio retardées des *idlt* secondes de son stocké. Il peut y avoir n'importe quel nombre d'unités *deltap* et/ou *deltapi* entre une paire lecture/écriture. Chacune reçoit un extrait audio sans changement de l'amplitude originale.

Ces opcodes peuvent fournir de multiples lectures de délai pour des lignes à retard arbitraires et des réseaux à rétroaction. Ils peuvent délivrer des délais constants ou variables, et sont utiles pour construire des effets de chorus, harmonizer et Doppler. Les délais constants (et certains de ceux qui varient lentement) ne nécessitent pas d'interpolation ; *deltap* leur convient très bien. Les délais variant à moyenne vitesse ou rapidement nécessiteront cependant les services supplémentaires de *deltapi*.

Les paires *delayr/delayw* peuvent être entrelacées. Pour associer une unité de lecture de délai à une unité *delayr*, elle doit non seulement être située entre ce *delayr* et le *delayw* approprié, mais aussi précéder tous les *delayr* suivants. Voir l'exemple 2. (Cette possibilité fut ajoutée dans la version 3.57 de Csound par Jens Groh et John ffitich).

*N.B.* Les délais de taux-k ne sont pas interpolés en interne, mais déroulent plutôt des décalages temporels d'échantillons audios ; c'est adéquat pour des délais changeant lentement. Cependant, pour les changements à vitesse moyenne ou rapides, il faut fournir en entrée des valeurs de délai avec une plus grande résolution de taux audio.

## Exemples

### Exemple 121. Exemple n°1 de deltap

```
asource buzz      1, 440, 20, 1
atime linseg      1, p3/2,.01, p3/2,1 ; trace a distance in secs
ampfac =          1/atime/atime        ; and calc an amp factor
adump delayr      1                   ; set maximum distance
amove deltapi      atime               ; move sound source past
```

```

delayw    asource      ; the listener
out       amove * amfac

```

## Exemple 122. Exemple n°2 de delayr

```

ainput1 = .....
ainput2 = .....
kdlyt1  = .....
kdlyt2  = .....

;Read delayed signal, first delayr instance:
adump    delayr 4.0
adly1    delayr kdlyt1      ; associated with first delayr instance

;Read delayed signal, second delayr instance:
adump    delayr 4.0
adly2    delayr kdlyt2      ; associated with second delayr instance

;Do some cross-coupled manipulation:
afdbk1 =      0.7 * adly1 + 0.7 * adly2 + ainput1
afdbk2 =     -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
delayw    afdbk1

;Feed back signal, associated with second delayr instance:
delayw    afdbk2
outs      adly1, adly2

```

## Voir Aussi

*delayr, delayrpi, delayrpn*

# deltapi

deltapi — Lit une ligne à retard avec des délais variables et interpolation.

## Description

Lit une ligne à retard avec des délais variables et interpolation.

## Syntaxe

```
ares deltapi xdl
```

## Exécution

*xdl* -- spécifie le délai de lecture en secondes. Chaque valeur est comprise entre 1 période de contrôle et le délai total de la paire lecture/écriture ; cependant, comme il n'y a pas de vérification interne du respect de cet intervalle, l'utilisateur est entièrement responsable. Chaque argument peut être une constante, une variable ou un signal variant dans le temps ; l'argument *xdl* de *deltapi* implique qu'une valeur de délai variant au taux audio est autorisée ici.

*deltapi* extrait le son par lecture interpolée. En interpolant entre deux échantillons voisins stockés *deltapi* restitue un délai particulier avec plus de précision, mais nécessite deux fois plus de temps de calcul.

Cet opcode peut lire directement dans une paire *delayr/delayw* en extrayant les données audio retardées des *idlt* secondes de son stocké. Il peut y avoir n'importe quel nombre d'unités *deltap* et/ou *deltapi* entre une paire lecture/écriture. Chacune reçoit un extrait audio sans changement de l'amplitude originale.

Ces opcodes peuvent fournir de multiples lectures de délai pour des lignes à retard arbitraires et des réseaux à rétroaction. Ils peuvent délivrer des délais constants ou variables, et sont utiles pour construire des effets de chorus, harmonizer et Doppler. Les délais constants (et certains de ceux qui varient lentement) ne nécessitent pas d'interpolation ; *deltap* leur convient très bien. Les délais variant à moyenne vitesse ou rapidement nécessiteront cependant les services supplémentaires de *deltapi*.

Les paires *delayr/delayw* peuvent être entrelacées. Pour associer une unité de lecture de délai à une unité *delayr*, elle doit non seulement être située entre ce *delayr* et le *delayw* approprié, mais aussi précéder tous les *delayr* suivants. Voir l'exemple 2. (Cette possibilité fut ajoutée dans la version 3.57 de Csound par Jens Groh et John ffitch).

*N.B.* Les délais de taux-k ne sont pas interpolés en interne, mais déroulent plutôt des décalages temporels d'échantillons audios ; c'est adéquat pour des délais changeant lentement. Cependant, pour les changements à vitesse moyenne ou rapides, il faut fournir en entrée des valeurs de délai avec une plus grande résolution de taux audio.

## Exemples

### Exemple 123. Exemple n°1 de deltap

```
asource buzz      1, 440, 20, 1
atime linseg    1, p3/2,.01, p3/2,1 ; trace a distance in secs
ampfac =         1/atime/atime      ; and calc an amp factor
```

```

adump    delayr  1                ; set maximum distance
amove    deltapi atime            ; move sound source past
          delayw asource          ; the listener
          out    amove * ampfac

```

## Exemple 124. Exemple n°2 de deltap

```

ainput1 = .....
ainput2 = .....
kdlyt1  = .....
kdlyt2  = .....

;Read delayed signal, first delayr instance:
adump    delayr  4.0
adly1    deltap  kdlyt1          ; associated with first delayr instance

;Read delayed signal, second delayr instance:
adump    delayr  4.0
adly2    deltap  kdlyt2          ; associated with second delayr instance

;Do some cross-coupled manipulation:
afdbk1 =      0.7 * adly1 + 0.7 * adly2 + ainput1
afdbk2 =     -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
          delayw afdbk1

;Feed back signal, associated with second delayr instance:
          delayw afdbk2
          outs   adly1, adly2

```

## Voir Aussi

*deltap, deltap3, deltapi*

# deltapn

deltapn — Lit une ligne à retard avec des délais variables.

## Description

Lit une ligne à retard avec des délais variables.

## Syntaxe

```
ares deltapn xnumsamps
```

## Exécution

*xnumsamps* -- spécifie le délai de lecture en secondes. Chaque valeur est comprise entre 1 période de contrôle et le délai total de la paire lecture/écriture ; cependant, comme il n'y a pas de vérification interne du respect de cet intervalle, l'utilisateur est entièrement responsable. Chaque argument peut être une constante, une variable ou un signal variant dans le temps.

*deltapn* est identique à *deltapi*, sauf que la durée du retard est exprimée en échantillons plutôt qu'en secondes (Hans Mikelson).

Cet opcode peut lire directement dans une paire *delayr/delayw* en extrayant les données audio retardées des *idlt* secondes de son stocké. Il peut y avoir n'importe quel nombre d'unités *deltap* et/ou *deltapi* entre une paire lecture/écriture. Chacune reçoit un extrait audio sans changement de l'amplitude originale.

Ces opcodes peuvent fournir de multiples lectures de délai pour des lignes à retard arbitraires et des réseaux à rétroaction. Ils peuvent délivrer des délais constants ou variables, et sont utiles pour construire des effets de chorus, harmonizer et Doppler. Les délais constants (et certains de ceux qui varient lentement) ne nécessitent pas d'interpolation ; *deltap* leur convient très bien. Les délais variant à moyenne vitesse ou rapidement nécessiteront cependant les services supplémentaires de *deltapi*.

Les paires *delayr/delayw* peuvent être entrelacées. Pour associer une unité de lecture de délai à une unité *delayr*, elle doit non seulement être située entre ce *delayr* et le *delayw* approprié, mais aussi précéder tous les *delayr* suivants. Voir l'exemple 2. (Cette possibilité fut ajoutée dans la version 3.57 de Csound par Jens Groh et John ffitich).

*N.B.* Les délais de taux-k ne sont pas interpolés en interne, mais déroulent plutôt des décalages temporels d'échantillons audios ; c'est adéquat pour des délais changeant lentement. Cependant, pour les changements à vitesse moyenne ou rapides, il faut fournir en entrée des valeurs de délai avec une plus grande résolution de taux audio.

## Exemples

### Exemple 125. Exemple n°1 de deltap

```
asource buzz      1, 440, 20, 1
atime linseg      1, p3/2,.01, p3/2,1 ; trace a distance in secs
ampfac =          1/atime/atime        ; and calc an amp factor
adump delayr      1                    ; set maximum distance
amove deltapi      atime                ; move sound source past
```



```

delayw    asource      ; the listener
out       amove * amfac

```

## Exemple 126. Exemple n°2 de deltap

```

ainput1 = .....
ainput2 = .....
kdlyt1  = .....
kdlyt2  = .....

;Read delayed signal, first delayr instance:
adump    delayr 4.0
adly1    deltap kdlyt1      ; associated with first delayr instance

;Read delayed signal, second delayr instance:
adump    delayr 4.0
adly2    deltap kdlyt2      ; associated with second delayr instance

;Do some cross-coupled manipulation:
afdbk1 =      0.7 * adly1 + 0.7 * adly2 + ainput1
afdbk2 =     -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
delayw   afdbk1

;Feed back signal, associated with second delayr instance:
delayw   afdbk2
outs     adly1, adly2

```

## Voir Aussi

*deltap, deltap3, deltapi*

# deltapx

deltapx — Lit depuis ou écrit dans une ligne à retard avec interpolation.

## Description

*deltapx* est semblable à *deltapi* ou à *deltap3*. Cependant, il permet une meilleure qualité d'interpolation. Cet opcode peut lire depuis et écrire dans une ligne à retard *delayr/delayw* avec interpolation.

## Syntaxe

```
aout deltapx adel, iwsiz
```

## Initialisation

*iwsiz* -- taille de la fenêtre d'interpolation en échantillons. Les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024. *iwsiz* = 4 utilise l'interpolation cubique. Des valeurs croissantes de *iwsiz* améliorent la qualité sonore au prix d'une utilisation plus intensive du CPU, et d'une durée de délai minimale.

## Exécution

*aout* -- Signal de sortie.

*adel* -- Délai en secondes.

```
a1      delayr    idlr
        deltapxw a2, adl1, iws1
a3      deltapx  adl2, iws2
        deltapxw a4, adl3, iws3
        delayw   a5
```

Durées de délai minimales et maximales :

$\text{idlr} \geq 1/\text{kr}$	Longueur de la ligne à retard
$\text{adl1} \geq (\text{iws1}/2)/\text{sr}$	Ecriture avant la lecture
$\text{adl1} \leq \text{idlr} - (1 + \text{iws1}/2)/\text{sr}$	(permet des délais plus courts)
$\text{adl2} \geq 1/\text{kr} + (\text{iws2}/2)/\text{sr}$	Temps de lecture
$\text{adl2} \leq \text{idlr} - (1 + \text{iws2}/2)/\text{sr}$	
$\text{adl2} \geq \text{adl1} + (\text{iws1} + \text{iws2}) / (2*\text{sr})$	
$\text{adl2} \geq 1/\text{kr} + \text{adl3} + (\text{iws2} + \text{iws3}) / (2*\text{sr})$	
$\text{adl3} \geq (\text{iws3}/2)/\text{sr}$	Ecriture après lecture
$\text{adl3} \leq \text{idlr} - (1 + \text{iws3}/2)/\text{sr}$	(permet une rétroaction)



### Note

Les tailles de fenêtres des autres opcodes que *deltapx* sont : *deltap*, *deltapn* : 1, *deltapi* : 2 (linéaire), *deltap3* : 4 (cubique).

## Exemples

```
a1      phasor      300.0
a1      = a1 - 0.5
a_      delayr      1.0
adel    phasor      4.0
adel    = sin (2.0 * 3.14159265 * adel) * 0.01 + 0.2
adel    deltapxw    a1, adel, 32
adel    phasor      2.0
adel    = sin (2.0 * 3.14159265 * adel) * 0.01 + 0.2
adel    deltapxw    a1, adel, 32
adel    = 0.3
a2      deltapx     adel, 32
a1      = 0
a1      delayw      a1
out      a2 * 20000.0
```

## Voir Aussi

*deltapxw*

## Crédits

Auteur : Istvan Varga  
Août 2001

Nouveau dans la version 4.13

# deltapxw

deltapxw — Mélange le signal d'entrée dans une ligne à retard.

## Description

*deltapxw* mélange le signal d'entrée dans une ligne à retard. Cet opcode peut être utilisé avec les unités de lecture (*deltap*, *deltapn*, *deltapi*, *deltap3* et *deltapx*) dans n'importe quel ordre ; la durée du délai étant la différence entre les dates de lecture et d'écriture. Cet opcode peut lire depuis et écrire dans une ligne à retard *delayr/delayw* avec interpolation.

## Syntaxe

**deltapxw** ain, adel, iwsiz

## Initialisation

*iwsiz* -- taille de la fenêtre d'interpolation en échantillons. Les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024. *iwsiz* = 4 utilise l'interpolation cubique. Des valeurs croissantes de *iwsiz* améliorent la qualité sonore au prix d'une utilisation plus intensive du CPU, et d'une durée de délai minimale.

## Exécution

*ain* -- Signal d'entrée.

*adel* -- Délai en secondes.

```
a1      delayr  idlr
          deltapxw a2, adl1, iws1
a3      deltapx  adl2, iws2
          deltapxw a4, adl3, iws3
          delayw  a5
```

Durées de délai minimales et maximales :

$idlr \geq 1/kr$	Longueur de la ligne à retard
$adl1 \geq (iws1/2)/sr$	Ecriture avant lecture
$adl1 \leq idlr - (1 + iws1/2)/sr$	(permet des délais plus courts)
$adl2 \geq 1/kr + (iws2/2)/sr$	Temps de lecture
$adl2 \leq idlr - (1 + iws2/2)/sr$	
$adl2 \geq adl1 + (iws1 + iws2) / (2*sr)$	
$adl2 \geq 1/kr + adl3 + (iws2 + iws3) / (2*sr)$	
$adl3 \geq (iws3/2)/sr$	Ecriture après lecture
$adl3 \leq idlr - (1 + iws3/2)/sr$	(permet une rétroaction)



## Note

Les tailles de fenêtres des autres opcodes que *deltapx* sont : *deltap*, *deltapn* : 1, *deltapi* : 2 (linéaire), *deltap3* : 4 (cubique).

## Exemples

```
a1      phasor      300.0
a1      = a1 - 0.5
a_      delayr      1.0
adel    phasor      4.0
adel    = sin (2.0 * 3.14159265 * adel) * 0.01 + 0.2
adel    deltapxw a1, adel, 32
adel    phasor      2.0
adel    = sin (2.0 * 3.14159265 * adel) * 0.01 + 0.2
adel    deltapxw a1, adel, 32
adel    = 0.3
a2      deltapx      adel, 32
a1      = 0
a1      delayw      a1
out      a2 * 20000.0
```

## Voir Aussi

*deltapx*

## Crédits

Auteur : Istvan Varga  
Août 2001

Nouveau dans la version 4.13

# denorm

denorm — Mixes low level noise to a list of a-rate signals

## Description

Mixes low level ( $\sim 1e-20$  for floats, and  $\sim 1e-56$  for doubles) noise to a list of a-rate signals. Can be used before IIR filters and reverbs to avoid denormalized numbers which may otherwise result in significantly increased CPU usage.

## Syntax

```
denorm a1[, a2[, a3[, ... ]]]
```

## Performance

*a1[, a2[, a3[, ... ]]]* -- signals to mix noise with

Some processor architectures (particularly Pentium IVs) are very slow at processing extremely small numbers. These small numbers can appear as a result of some decaying feedback process like reverb and IIR filters. Low level noise can be added so that very small numbers are never reached, and they are 'absorbed' by this 'noise floor'.

If CPU usage goes to 100% at the end of reverb tails, or you get audio glitches in processes that shouldn't use too much CPU, using *denorm* before the culprit opcode or process might solve the problem.

## Credits

Author: Istvan Varga  
2005

# diff

diff — Modify a signal by differentiation.

## Description

Modify a signal by differentiation.

## Syntax

```
ares diff asig [, iskip]
```

```
kres diff ksig [, iskip]
```

## Initialization

*iskip* (optional) -- initial disposition of internal save space (see *reson*). The default value is 0.

## Performance

*integ* and *diff* perform integration and differentiation on an input control signal or audio signal. Each is the converse of the other, and applying both will reconstruct the original signal. Since these units are special cases of low-pass and high-pass filters, they produce a scaled (and phase shifted) output that is frequency-dependent. Thus *diff* of a sine produces a cosine, with amplitude  $2 * \sin(\pi * Hz / sr)$  that of the original (for each component partial); *integ* will inversely affect the magnitudes of its component inputs. With this understanding, these units can provide useful signal modification.

## Examples

Here is an example of the diff opcode. It uses the file *diff.csd* [examples/diff.csd].

### Exemple 127. Example of the diff opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o diff.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 -- a normal instrument.
instr 1
; Generate a band-limited pulse train.
```

```

asrc buzz 20000, 440, 20, 1

out asrc
endin

; Instrument #2 -- a differentiated instrument.
instr 2
; Generate a band-limited pulse train.
asrc buzz 20000, 440, 20, 1

; Emphasize the highs.
al diff asrc

out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>

```

## See Also

*downsamp, integ, interp, samphold, upsamp*

## Credits

Example written by Kevin Conder.



# diskgrain

diskgrain — Synthèse granulaire synchrone, utilisant un fichier son comme source.

## Description

*diskgrain* implémente la synthèse granulaire synchrone. La source sonore des grains est obtenue en lisant un fichier son contenant les échantillons de la forme d'onde source.

## Syntaxe

```
asig diskgrain Sfilename, kamp, kfreq, kpitch, kgrsize, kprate, \  
      ifun, iolaps [,imaxgrsize , ioffset]
```

## Initialisation

*Sfilename* -- fichier son source.

*ifun* -- table de fonction de l'enveloppe de grain.

*iolaps* -- nombre maximum de chevauchements,  $\max(kfreq) * \max(kgrsize)$ . Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

*imaxgrsize* -- taille de grain maximale en secondes (par défaut 1.0).

*ioffset* -- décalage initial en secondes à partir du début du fichier (par défaut 0).

## Exécution

*kamp* -- pondération de l'amplitude

*kfreq* -- fréquence de génération des grains, ou densité, en grains/sec.

*kpitch* -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers)

*kgrsize* -- taille de grain en secondes.

*kprate* -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

Le générateur de grain contrôle complètement la fréquence (grains/sec), l'amplitude globale, la hauteur de grain (un incrément de l'échantillonnage) et la taille de grain (en secondes), comme paramètres fixes ou variant dans le temps (signaux). La vitesse du pointeur de grain est un paramètre supplémentaire qui contrôle la position à laquelle le générateur commencera à lire les échantillons dans le fichier pour chaque grain successif. Elle est mesurée en fraction de la taille de grain, si bien qu'une valeur de 1 (par défaut) provoquera la lecture de chaque grain successif à partir de l'endroit où le grain précédent s'est terminé. Avec une valeur de 0.5 le grain suivant commencera à la position médiane entre le début et la fin du grain précédent, etc... Avec une valeur de 0 le générateur lit toujours à partir d'une position fixe (quelque soit l'endroit où il se trouvait précédemment). Une valeur négative décrémentera les positions

du pointeur. Ce contrôle donne plus de flexibilité pour créer des modifications de l'échelle temporelle pendant la resynthèse.

*Diskgrain* générera n'importe quel nombre de flux de grain parallèles (en fonction de la densité/fréquence de grain) borné par la valeur de *iolaps* (par défaut 100). Le nombre de flux (grains se chevauchant) est déterminé par  $kgrsize * kfreq$ . Plus il y aura de chevauchements, plus il y aura de calculs ce qui pourra empêcher la synthèse en temps réel (selon la puissance du processeur).

*Diskgrain* peut simuler une synthèse formantique à la FOF, si on emploie une forme adéquate comme enveloppe de grain et une forme d'onde sinus comme onde de grain. Pour cette utilisation, on peut choisir des tailles de grain d'environ 0.04 secondes. La fréquence centrale du formant est déterminée par la hauteur de grain. Comme celle-ci est exprimée en incrément d'échantillonnage, il faut pondérer cette valeur par  $tablesiz/sr$  pour obtenir une fréquence en Hz. La fréquence de grain déterminera le fondamental.

Cet opcode est une variation sur l'opcode *syncgrain*.

## Exemples

Voici un exemple de l'opcode *diskgrain*. Il utilise le fichier *diskgrain.csd* [examples/diskgrain.csd].

### Exemple 128. Exemple de l'opcode *diskgrain*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 128

instr 1
iolaps = 2
igrsize = 0.04
ifreq = iolaps/igrsize
ips = 1/iolaps

istr = p4 /* timescale */
ipitch = p5 /* pitchscale */

a1 diskgrain "mary.wav", 32000, ifreq, ipitch, igrsize, ips*istr, 1, iolaps

out a1
endin

</CsInstruments>
<CsScore>
f 1 0 8192 20 2 1 ;Hanning function

;          timescale  pitchscale
i 1 0 5 1 1
i 1 + 5 2 1
i 1 + 5 1 0.75
i 1 + 5 1.5 1.5
i 1 + 5 0.5 1.5

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Mai 2007  
Nouveau dans Csound 5.06

# diskin

diskin — Obsolète. Lit des données audio d'un périphérique ou d'un flot externe et peut altérer leur hauteur.

## Description

Obsolète. Lit des données audio d'un périphérique ou d'un flot externe et peut altérer leur hauteur.

## Syntaxe

```
ar1 [, ar2 [, ar3 [, ... ar24]]] diskin ifilcod, kpitch [, iskptim] \  
[, iwraparound] [, iformat] [, iskipinit]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son source. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) puis par SFDIR.

*iskptim* (facultatif) -- portion du son en entrée à ignorer, exprimée en secondes. La valeur par défaut est 0.

*iformat* (facultatif) -- spécifie le format des données audio du fichier :

- 1 = caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 2 = octets sur 8 bit A-law
- 3 = octets sur 8 bit U-law
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers non signés sur 8 bit (non disponible dans les versions de Csound antérieures à la 5.00)
- 8 = entiers sur 24 bit (non disponible dans les versions de Csound antérieures à la 5.00)
- 9 = doubles sur 64 bit (non disponible dans les versions de Csound antérieures à la 5.00)

*iwraparound* -- 1 = activé, 0 = désactivé (parcours cyclique du fichier dans les deux directions)

*iskipinit* -- supprime toute initialisation s'il est non nul (vaut 0 par défaut). Fut introduit dans la version 4\_23f13 et dans csound5.

Si *iformat* = 0, il est déduit de l'en-tête du fichier, et s'il n'y a pas d'en-tête, de l'option de ligne de commande *-o* de Csound. La valeur par défaut est 0.

## Exécution



### Note

*diskin* est obsolète car il plante facilement dans certaines circonstances. Utiliser plutôt *diskin2*.

*kpitch* -- N'importe quel nombre réel. Un nombre négatif signifie une restitution à l'envers. Ce nombre est un rapport de hauteur où :

- 1 = hauteur normale
- 2 = 1 octave plus haut
- 3 = 12ème plus haut, etc.
- .5 = 1 octave plus bas
- .25 = 2 octaves plus bas, etc.
- -1 = hauteur normale à l'envers
- -2 = 1 octave plus haut à l'envers, etc.

*diskin* est semblable à *soundin* sauf qu'il peut modifier la hauteur du son qui est lu.



### Note pour les utilisateurs de Windows

Les utilisateurs de Windows utilisent normalement des anti-slash, « \ », pour spécifier les chemins de leurs fichiers. Par exemple un utilisateur de Windows pourra utiliser le chemin « c:\music\samples\loop001.wav ». Ceci pose problème car les anti-slash servent habituellement à spécifier des caractères spéciaux.

Pour spécifier correctement ce chemin dans Csound on peut utiliser :

- soit le *slash* : c:/music/samples/loop001.wav
- soit le *caractère spécial d'anti-slash*, « \\ » : c:\\music\\samples\\loop001.wav

## Exemples

Voici un exemple de l'opcode *diskin*. Il utilise les fichiers *diskin.csd* [examples/diskin.csd] et *beats.wav* [examples/beats.wav].

### Exemple 129. Exemple de l'opcode *diskin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;;RT audio I/O
; -o disk.in.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
; Play the audio file
asig disk.in "beats.wav", 1
out asig
endin

; Instrument #2 - play an audio file backwards.
instr 2
; Play the audio file backwards.
asig disk.in "beats.wav", -1
out asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
i 2 3 3
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir Aussi

*in, inh, ino, inq, ins, soundin* and *diskin2*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

Nouveau dans la version 3.46

Exemple écrit par Kevin Conder.

Avertissement pour les utilisateurs de Windows ajouté par Kevin Conder, avril 2002

# diskin2

diskin2 — Lit des données audio depuis un fichier, et peut altérer leur hauteur au moyen d'un des types d'interpolation disponibles ainsi que convertir le taux d'échantillonnage pour s'accorder à celui de l'orchestre.

## Description

Lit des données audio depuis un fichier, et peut altérer leur hauteur au moyen d'un des types d'interpolation disponibles ainsi que convertir le taux d'échantillonnage pour s'accorder à celui de l'orchestre. *diskin2* peut également lire des fichiers multicanaux dont le nombre de canaux est compris entre 1 et 24. *diskin2* offre plus de contrôle et une meilleure qualité de son que *diskin* mais au prix d'une utilisation plus intensive des ressources CPU.

## Syntaxe

```
a1[, a2[, ... a24]] diskin2 ifilcod, kpitch[, iskiptim \
[, iwrap[, iformat [, iwsz[, ibufsz[, iskipinit]]]]]]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son source. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) puis par SFDIR. Voir aussi *GEN01*. Note : il est possible que les fichiers contenant plus de  $2^{31}-1$  trames d'échantillons ne soient pas joués correctement sur les plates-formes 32 bit ; cela donne une longueur maximale d'environ trois heures avec un taux d'échantillonnage de 192000 Hz.

*iskiptim* (facultatif, zéro par défaut) -- portion du son en entrée à ignorer, exprimée en secondes, en supposant que *kpitch*=1. Peut être négatif, pour ajouter *-iskiptim/kpitch* secondes de délai au lieu de d'ignorer une partie du son.



### Note

Si *iwrap* est différent de 0 (lecture cyclique), *iskiptim* ne retardera pas le son si une valeur négative est utilisée. Au lieu de cela, la lecture commencera du même décalage avant la fin du fichier.

*iwrap* (facultatif, zéro par défaut) -- s'il a n'importe quelle valeur non nulle, les positions de lecture négatives ou au-delà de la fin du fichier sont ramenées à l'intérieur de la durée du fichier son au lieu d'être remplacées par des échantillons nuls. Pratique pour jouer un fichier en boucle.



### Note

Si *iwrap* est activé, la longueur du fichier ne doit pas être inférieure à la taille de la fenêtre d'interpolation (voir ci-dessous), sinon il pourra y avoir des craquements dans le son de sortie.

*iformat* (facultatif, zéro par défaut) -- format d'échantillon, seulement pour les fichiers bruts (sans en-

tête). Ce paramètre est ignoré si le fichier a un en-tête. Les valeurs admises sont :

- 0 : entiers courts sur 16 bit
- 1 : caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 2 : octets sur 8 bit A-law
- 3 : octets sur 8 bit U-law
- 4 : entiers courts sur 16 bit
- 5 : entiers longs sur 32 bit
- 6 : flottants sur 32 bit
- 7 : entiers non signés sur 8 bit
- 8 : entiers sur 24 bit
- 9 : doubles sur 64 bit

*isize* (facultatif, zéro par défaut) -- taille de la fenêtre d'interpolation, en échantillons. Peut prendre une de ces valeurs :

- 1 : arrondi à l'échantillon le plus proche (pas d'interpolation, pour *kpitch*=1)
- 2 : interpolation linéaire
- 4 : interpolation cubique
- >= 8 : interpolation par sinc de *isize* points avec anti-aliasing (lent)

Zéro ou des valeurs négatives sélectionnent la valeur par défaut, qui est l'interpolation cubique.



## Note

S'il y a interpolation, *kpitch* est automatiquement mis à l'échelle par le rapport des taux d'échantillonnage du fichier et de l'orchestre, afin que le fichier soit toujours joué à la hauteur originale si *kpitch* vaut 1. Cependant, la conversion du taux d'échantillonnage est désactivée lorsque *isize* vaut 1.

*ibufsize* (facultatif, zéro par défaut) -- taille du tampon en échantillons mono (pas en trames d'échantillons). Ce n'est que la valeur suggérée, la valeur retenue étant arrondie afin que le nombre de trames d'échantillons soit une puissance entière de deux et soit comprise entre 128 (ou *isize* s'il est supérieur à 128) et 1048576. La valeur par défaut, 4096, choisie par zéro ou une valeur négative, sera adéquate dans la plupart des cas, mais lors du mélange de plusieurs fichiers son de grande taille en temps différé, une grande taille de tampon est recommandée pour améliorer l'efficacité des lectures sur disque. Pour une sortie en temps réel, la lecture des fichiers depuis un RAM disque rapide (sur les plates-formes qui le permettent) avec une petite taille de tampon est préférable.

*iskipinit* (facultatif, zéro par défaut) -- supprime l'initialisation s'il est non nul.

## Exécution

*a1* ... *a24* -- signaux de sortie, dans l'intervalle allant de -0dbfs à 0dbfs. Tous les échantillons avant le



début du fichier (positions négatives) et après la fin du fichier prennent la valeur zéro, à moins que *iwrap* soit non nul. Le nombre d'arguments de sortie doit être le même que le nombre de canaux du fichier son - lequel peut être déterminé avec l'opcode *filenchnls*, sinon il y aura une erreur d'initialisation.



## Note

Il est plus efficace de lire un seul fichier avec plusieurs canaux, que plusieurs fichiers à un seul canal, spécialement avec de grandes valeurs de *iwsiz*e.

*kpitch* -- transpose la hauteur du son en entrée par ce facteur (par exemple 0.5 signifie une octave plus bas, 2 une octave plus haut, et 1 la hauteur originale). Des valeurs fractionnaires et négatives sont permises (les dernières provoquant une lecture à l'envers, cependant, dans ce cas, *iskiptim* doit prendre une valeur positive, par exemple la longueur du fichier, ou bien *iwrap* doit être non nul, sinon rien ne sera joué). S'il y a interpolation et que le taux d'échantillonnage du fichier est différent de celui de l'orchestre, le rapport de transposition est automatiquement ajusté de façon à ce que *kpitch*=1 joue à la hauteur originale. Un *iwsiz*e élevé (40 ou plus) peut améliorer significativement la qualité du son lors d'une transposition vers l'aigu, au prix d'une utilisation plus intensive des ressources CPU.

## Exemple

```
<CsSoundSynthesizer>
<CsOptions>
; set this to a directory where beats.wav can be found
--env:SSDIR+=/Csound/Documentation/manual/examples
</CsOptions>
<CsInstruments>
sr      = 48000
ksmps   = 32
nchnls  = 2

        instr 1

ktrans  linseg 1, 5, 2, 10, -2
al      diskin2 "beats.wav", ktrans, 0, 1, 0, 32
        outs al, al
        endin

</CsInstruments>
<CsScore>

i 1 0 15
e

</CsScore>
</CsSoundSynthesizer>
```

## Voir Aussi

*in*, *inh*, *ino*, *inq*, *ins*, *soundin* and *diskin*

## Crédits

Auteur : Istvan Varga  
2005

Nouveau dans la version 5.00

# dispfft

displayfft — Affiche la transformée de Fourier d'un signal audio ou de contrôle.

## Description

Ces unités affichent les valeurs d'initialisation de l'orchestre ou produisent un affichage graphique de signaux de contrôle ou audio de l'orchestre. Des fenêtres X11 sont utilisées s'il est activé, sinon (ou si l'option `-g` est positionnée) on a un affichage approximatif en caractères ASCII.

## Syntaxe

```
dispfft xsig, iprd, iwsiz [, iwtyp] [, idbout] [, iwtflg]
```

## Initialisation

*iprd* -- la période d'affichage en secondes.

*iwsiz* -- taille de la fenêtre d'entrée en échantillons. Une fenêtre de *iwsiz* points produira une transformée de Fourier de *iwsiz*/2 points, répartis linéairement en fréquence de 0 à *sr*/2. *iwsiz* doit être une puissance de 2, comprise entre 16 et 4096. Les fenêtres peuvent se chevaucher.

*iwtyp* (facultatif, 0 par défaut) -- type de fenêtre. 0 = rectangulaire, 1 = Hanning. La valeur par défaut est 0 (rectangulaire).

*idbout* (facultatif, 0 par défaut) -- unité d'affichage des coefficients de Fourier. 0 = magnitude, 1 = décibels. La valeur par défaut est 0 (magnitude).

*iwtflg* (facultatif, 0 par défaut) -- indicateur de maintien. S'il est différent de zéro, chaque affichage est maintenu jusqu'à ce que l'utilisateur le libère. La valeur par défaut est 0 (pas de maintien).

## Exécution

*dispfft* -- affiche la transformée de Fourier d'un signal audio ou de contrôle (*asig* ou *ksig*) chaque *iprd* secondes au moyen de la méthode de transformée de Fourier rapide.

## Exemples

Voici un exemple de l'opcode *dispfft*. Il utilise les fichiers *dispfft.csd* [examples/dispfft.csd] et *beats.wav* [examples/beats.wav].

### Exemple 130. Exemple de l'opcode *dispfft*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o dispfft.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  asig soundin "beats.wav"
  dispfft asig, 1, 512
  out asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*display, print*

## Crédits

Exemple écrit par Kevin Conder.

# display

`display` — Affiche un signal audio ou de contrôle sur un graphique amplitude/temps.

## Description

Ces unités affichent les valeurs d'initialisation de l'orchestre ou produisent un affichage graphique de signaux de contrôle ou audio de l'orchestre. Des fenêtres X11 sont utilisées s'il est activé, sinon (ou si l'option `-g` est positionnée) on a un affichage approximatif en caractères ASCII.

## Syntaxe

```
display xsig, iprd [, inprds] [, iwtflg]
```

## Initialisation

*iprd* -- la période d'affichage en secondes.

*inprds* (facultatif, 1 par défaut) -- Nombre de périodes d'affichage retenues dans chaque graphique. Les valeurs supérieures ou égales à 2 donneront une perspective plus étendue du mouvement du signal. La valeur par défaut est 1 (chaque graphique est entièrement renouvelé). *inprds* est un facteur d'échelle pour la forme d'onde affichée, qui contrôle combien de trames d'échantillon de longueur *iprd* sont dessinées dans la fenêtre (la valeur par défaut qui est aussi la valeur minimale est 1.0). Des valeurs supérieures de *inprds* provoquent un dessin plus lent (plus de points à dessiner) mais feront défiler la forme d'onde à travers la fenêtre, ce qui est utile avec de faibles valeurs de *iprd*.

*iwtflg* (facultatif, 0 par défaut) -- indicateur de maintien. S'il est différent de zéro, chaque affichage est maintenu jusqu'à ce que l'utilisateur le libère. La valeur par défaut est 0 (pas de maintien).

## Exécution

`display` -- affiche le signal audio ou de contrôle *xsig* chaque *iprd* secondes, sur un graphique amplitude/temps.

## Exemples

Voici un exemple de l'opcode `display`. Il utilise le fichier `display.csd` [examples/display.csd].

### Exemple 131. Exemple de l'opcode `display`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o display.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Go from 1000 to 0 linearly, over the period defined by p3.
klin line 1000, p3, 0

; Create a new display each second, wait for the user.
display klin, 1, 1, 1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*dispfft, print*

## Crédits

Commentaires sur le paramètre *inprds* par Rasmus Ekman.

Exemple écrit par Kevin Conder.

# distort

distort — Distort an audio signal via waveshaping and optional clipping.

## Description

## Syntax

```
ar distort asig, kdist, ifn[, ihp, istor]
```

## Initialization

*ifn* -- table number of a waveshaping function with extended guard point. The function can be of any shape, but it should pass through 0 with positive slope at the table mid-point. The table size need not be large, since it is read with interpolation.

*ihp* -- (optional) half-power point (in cps) of an internal low-pass filter. The default value is 10.

*istor* -- (optional) initial disposition of internal data space (see *reson*). The default value is 0.

## Performance

*asig* -- Audio signal to be processed

*kdist* -- Amount of distortion (usually between 0 and 1)

This unit distorts an incoming signal using a waveshaping function *ifn* and a distortion index *kdist*. The input signal is first compressed using a running rms, then passed through a waveshaping function which may modify its shape and spectrum. Finally it is rescaled to approximately its original power.

The amount of distortion depends on the nature of the shaping function and on the value of *kdist*, which generally ranges from 0 to 1. For low values of *kdist*, we should like the shaping function to pass the signal almost unchanged. This will be the case if, at the mid-point of the table, the shaping function is near-linear and is passing through 0 with positive slope. A line function from -1 to +1 will satisfy this requirement; so too will a sigmoid (sinusoid from 270 to 90 degrees). As *kdist* is increased, the compressed signal is expanded to encounter more and more of the shaping function, and if this becomes non-linear the signal is increasingly *bent* on read-through to cause distortion.

When *kdist* becomes large enough, the read-through process will eventually hit the outer limits of the table. The table is not read with wrap-around, but will 'stick' at the end-points as the incoming signal exceeds them; this introduces clipping, an additional form of signal distortion. The point at which clipping begins will depend on the complexity (rms-to-peak value) of the input signal. For a pure sinusoid, clipping will begin only as *kdist* exceeds 0.7; for a more complex input, clipping might begin at a *kdist* of 0.5 or much less. *kdist* can exceed the clip point by any amount, and may be greater than 1.

The shaping function can be made arbitrarily complex for extra effect. It should generally be continuous, though this is not a requirement. It should also be well-behaved near the mid-point, and roughly balanced positive-negative overall, else some excessive DC offset may result. The user might experiment with more aggressive functions to suit the purpose. A generally positive slope allows the distorted signal to be mixed with the source without phase cancellation.

*distort* is useful as an effects process, and is usually combined with reverb and chorusing on effects

busses. However, it can alternatively be used to good effect within a single instrument.

## Examples

```
gifn  ftgen      0,0, 257, 9, .5,1,270      ; define a sigmoid, or better
gifn  ftgen      0,0, 257, 9, .5,1,270,1.5,.33,90,2.5,.2,270,3.5,.143,90,4.5,.111,270

kdist  line      0, 10, 1.2                ; and over 10 seconds
aout   distort    asig, kdist, gifn         ; gradually increase the distortion
```

## Credits

Written by Barry L. Vercoe for Extended Csound and released in csound5.

# distort1

distort1 — Modified hyperbolic tangent distortion.

## Description

Implementation of modified hyperbolic tangent distortion. *distort1* can be used to generate wave shaping distortion based on a modification of the *tanh* function.

$$\text{aout} = \frac{\exp(\text{asig} * (\text{shape1} + \text{pregain})) - \exp(\text{asig} * (\text{shape2} - \text{pregain}))}{\exp(\text{asig} * \text{pregain}) + \exp(-\text{asig} * \text{pregain})}$$

## Syntax

ares **distort1** asig, kpregain, kpostgain, kshape1, kshape2[, imode]

## Initialization

*imode* (Csound version 5.00 and later only; optional, defaults to 0) -- scales kpregain, kpostgain, kshape1, and kshape2 for use with audio signals in the range -32768 to 32768 (imode=0), -0dbfs to 0dbfs (imode=1), or disables scaling of kpregain and kpostgain and scales kshape1 by kpregain and kshape2 by -kpregain (imode=2).

## Performance

*asig* -- is the input signal.

*kpregain* -- determines the amount of gain applied to the signal before waveshaping. A value of 1 gives slight distortion.

*kpostgain* -- determines the amount of gain applied to the signal after waveshaping.

*kshape1* -- determines the shape of the positive part of the curve. A value of 0 gives a flat clip, small positive values give sloped shaping.

*kshape2* -- determines the shape of the negative part of the curve.

## Examples

Here is an example of the distort1 opcode. It uses the file *distort1.csd* [examples/distort1.csd].

### Exemple 132. Example of the distort1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o distort1.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

gadist init 0

instr 1
  iamp = p4
  ifqc = cpspch(p5)
  asig pluck iamp, ifqc, ifqc, 0, 1
  gadist = gadist + asig
endin

instr 50
  kpre init p4
  kpost init p5
  kshap1 init p6
  kshap2 init p7
  aout distort1 gadist, kpre, kpost, kshap1, kshap2

  outs aout, aout

  gadist = 0
endin

</CsInstruments>
<CsScore>

; Sta Dur Amp Pitch
i1 0.0 3.0 10000 6.00
i1 0.5 2.5 10000 7.00
i1 1.0 2.0 10000 7.07
i1 1.5 1.5 10000 8.00

; Sta Dur PreGain PostGain Shape1 Shape2
i50 0 3 2 1 0 0
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Hans Mikelson  
December 1998

New in Csound version 3.50

# divz

divz — Division protégée de deux nombres.

## Syntaxe

```
ares divz xa, xb, ksubst
```

```
ires divz ia, ib, isubst
```

```
kres divz ka, kb, ksubst
```

## Description

Division protégée de deux nombres.

## Initialisation

Lorsque  $b$  est différent de zéro, le résultat reçoit la valeur de  $a / b$  ; si  $b$  est égal à zéro, le résultat prend la valeur de *subst*.

## Exemples

Voici un exemple de l'opcode divz. Il utilise le fichier *divz.csd* [examples/divz.csd].

### Exemple 133. Exemple de l'opcode divz.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o divz.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Define the numbers to be divided.
ka init 200
; Linearly change the value of kb from 200 to 0.
kb line 0, p3, 200
; If a "divide by zero" error occurs, substitute -1.
ksubst init -1

; Safely divide the numbers.
kresults divz ka, kb, ksubst
```

```
; Print out the results.
printks "%f / %f = %f\\n", 0.1, ka, kb, kresults
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
200.000000 / 0.000000 = -1.000000
200.000000 / 19.999887 = 10.000056
200.000000 / 40.000027 = 4.999997
```

## Voir Aussi

*=, init, tival*

## Crédits

Auteur : John ffitch d'après une idée de Barry L. Vercoe

Exemple écrit par Kevin Conder.

# doppler

doppler — A fast and robust method for approximating sound propagation, achieving convincing Doppler shifts without having to solve equations.

## Description

A fast and robust method for approximating sound propagation, achieving convincing Doppler shifts without having to solve equations. The method computes frequency shifts based on reading an input delay line at a delay time computed from the distance between source and mic and the speed of sound. One instance of the opcode is required for each dimension of space through which the sound source moves. If the source sound moves at a constant speed from in front of the microphone, through the microphone, to behind the microphone, then the output will be frequency shifted above the source frequency at a constant frequency while the source approaches, then discontinuously will be shifted below the source frequency at a constant frequency as the source recedes from the microphone. If the source sound moves at a constant speed through a point to one side of the microphone, then the rate of change of position will not be constant, and the familiar Doppler frequency shift typical of a siren or engine approaching and receding along a road beside a listener will be heard.

## Syntax

```
ashifted doppler asource, ksourceposition, kmicposition [, isoundspeed, ifiltercutoff]
```

## Initialization

*isoundspeed* (optional, default=340.29) -- Speed of sound in meters/second.

*ifiltercutoff* (optional, default=6) -- Rate of updating the position smoothing filter, in cycles/second.

## Performance

*asource* -- Input signal at the sound source.

*ksourceposition* -- Position of the source sound in meters. The distance between source and mic should not be changed faster than about 3/4 the speed of sound.

*kmicposition* -- Position of the recording microphone in meters. The distance between source and mic should not be changed faster than about 3/4 the speed of sound.

## Examples

Here is an example of the doppler opcode. It uses the file *doppler.csd* [examples/doppler.csd].

### Exemple 134. Example of the doppler opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>  
<CsOptions>  
-RWdo doppler.wav
```

```

</CsOptions>
<CsInstruments>
sr      = 44100
ksmps  = 20
nchnls = 1

iattack  instr 1 0.05
irelease  init   0.05
isustain  init   p3
p3        init   iattack + isustain + irelease
kdamping  linseg  0.0, iattack, 1.0, isustain, 1.0, irelease, 0.0
kmic      init   4
          ; Position envelope, with a changing rate of change of position.
kposition transeg 4, 4, 0, 120, 4, 4, -3, 50, 2, 4, 0
ismoothinghz init 6
ispeedofsound init 340.29
asignal    vco2  10000, 440
aoutput    doppler asignal, kposition, kmic, ispeedofsound, ismoothinghz
          out      aoutput * kdamping
          endin
</CsInstruments>
<CsScore>
i 1 1.0 20.0
e 1
</CsScore>
</CsoundSynthesizer>

```

## Credits

Author of algorithm: Peter Brinkmann  
 Author of opcode: Michael Gogins  
 January 2010

New in Csound version 5.11

# downsamp

downsamp — Modify a signal by down-sampling.

## Description

Modify a signal by down-sampling.

## Syntax

```
kres downsamp asig [, iwlen]
```

## Initialization

*iwlen* (optional) -- window length in samples over which the audio signal is averaged to determine a downsampled value. Maximum length is *ksmps*; 0 and 1 imply no window averaging. The default value is 0.

## Performance

*downsamp* converts an audio signal to a control signal by downsampling. It produces one kval for each audio control period. The optional window invokes a simple averaging process to suppress foldover.

## Examples

Here is an example of the *downsamp* opcode. It uses the file *downsamp.csd* [examples/downsamp.csd].

### Exemple 135. Example of the *downsamp* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o downsamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create a noise signal at a-rate.
anoise noise 20000, 0.2

; Downsample the noise signal to k-rate.
knoise downsamp anoise
```

```
; Use the noise signal at k-rate.
a1 oscil 30000, knoise, 1
out anoise
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*diff, integ, interp, samphold, upsamp*

## Credits

Example written by Kevin Conder.

# dripwater

dripwater — Modèle semi-physique d'une goutte d'eau.

## Description

*dripwater* est un modèle semi-physique d'une goutte d'eau. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntax

```
ares dripwater kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \  
    [, ifreq1] [, ifreq2]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 10.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$\text{damping\_amount} = 0,996 + (\text{idamp} * 0,002)$

La valeur par défaut de *damping\_amount* est 0,996 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 2,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale. Rasmus Ekman proposee un intervalle de 1,4 à 1,75. Il suggère aussi une valeur maximale de 1,9 au lieu de la limite théorique de 2,0.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 450.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 600.

*ifreq2* (facultatif) -- La seconde fréquence de résonance. La valeur par défaut est 750.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode *dripwater*. Il utilise le fichier *dripwater.csd* [examples/dripwater.csd].



## Exemple 136. Exemple de l'opcode dripwater.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dripwater.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 01 ;example of a water drip
a1 line 5, p3, 5 ;preset an amplitude boost
a2 dripwater p4, 0.01, 0, .9 ;dripwater needs a little amplitude help at these values
a3 product a1, a2 ;increase amplitude
out a3
endin

</CsInstruments>
<CsScore>

i1 0 1 20000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*bamboo, guiro, sleighbells, tambourine*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# dssiactivate

dssiactivate — Activates or deactivates a DSSI or LADSPA plugin.

## Syntax

```
dssiactivate ihandle, ktoggle
```

## Description

*dssiactivate* is used to activate or deactivate a DSSI or LADSPA plugin. It calls the plugin's `activate()` and `deactivate()` functions if they are provided.

## Initialization

*ihandle* - the number which identifies the plugin, generated by *dssiinit*.

## Performance

*ktoggle* - Selects between activation (*ktoggle*=1) and deactivation (*ktoggle*=0).

*dssiactivate* is used to turn on and off plugins if they provide this facility. This may help conserve CPU processing in some cases. For consistency, all plugins must be activated to produce sound. An inactive plugin produces silence.

Depending on the plugin's implementation, this may cause interruptions in the realtime audio process, so use with caution.

*dssiactivate* may cause audio stream breakups when used in realtime, so it is recommended to load all plugins to be used before playing.



### Avertissement

Please note that even if `activate()` and `deactivate()` functions are not present in a plugin, *dssiactivate* must be called for the plugin to produce sound.

## Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

# dssiaudio

dssiaudio — Processes audio using a LADSPA or DSSI plugin.

## Syntax

```
aout1 [, aout2, aout3, aout4] dssiaudio ihandle, ain1 [,ain2, ain3, ain4]
```

## Description

*dssiaudio* generates audio by processing an input signal through a LADSPA plugin.

## Initialization

*ihandle* - handle for the plugin returned by *dssiinit*

## Performance

*aout1, aout2, etc* - Audio output generated by the plugin

*ain1, ain2, etc* - Audio provided to the plugin for processing

*dssiaudio* runs a plugin on the provided audio and produces audio output. Currently upto four inputs and outputs are provided. You should provide signal for all the plugins audio inputs, otherwise unpredictable results may occur. If the plugin doesn't have any input (e.g Noise generator) you must still provide at least one input variable, which will be ignored with a message.

Only one *dssiaudio* should be executed once per plugin, or strange results may occur.

## Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

# dssictls

dssictls — Send control information to a LADSPA or DSSI plugin.

## Syntax

```
dssictls ihandle, iport, kvalue, ktrigger
```

## Description

*dssictls* sends control values to a plugin's control port

## Initialization

*ihandle* - handle for the plugin returned by *dssiinit*

*iport* - control port number

## Performance

*kvalue* - value to be assigned to the port

*ktrigger* - determines whether the control information will be sent (*ktrigger* = 1) or not. This is useful for thinning control information, generating *ktrigger* with *metro*

*dssictls* sends control information to a LADSPA or DSSI plugin's control port. The valid control ports and ranges are given by *dssiinit*. Using values outside the ranges may produce unspecified behaviour.

## Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

# dssiinit

dssiinit — Loads a DSSI or LADSPA plugin.

## Syntax

```
ihandle dssiinit ilibraryname, ipluginindex [, iverbose]
```

## Description

*dssiinit* is used to load a DSSI or LADSPA plugin into memory for use with the other dssi4cs opcodes. Both LADSPA effects and DSSI instruments can be used.

## Initialization

*ihandle* - the number which identifies the plugin, to be passed to other dssi4cs opcodes.

*ilibraryname* - the name of the .so (shared object) file to load.

*ipluginindex* - The index of the plugin to be used.

*iverbose* (optional) - show plugin information and parameters when loading. (default = 1)

*dssiinit* looks for *ilibraryname* on LADSPA\_PATH and DSSI\_PATH. One of these variables must be set, otherwise *dssiinit* will return an error. LADSPA and DSSI libraries may contain more than one plugin which must be referenced by its index. *dssiinit* then attempts to find plugin index *ipluginindex* in the library and load the plugin into memory if it is found. To find out which plugins you have available and their index numbers you can use: *dssilist*.

If *iverbose* is not 0 (the default), information about the plugin detailing its characteristics and its ports will be shown. This information is important for opcodes like *dssictls*.

Plugins are set to inactive by default, so you *\*must\** use *dssiactivate* to get the plugin to produce sound. This is required even if the plugin doesn't provide an activate() function.

*dssiinit* may cause audio stream breakups when used in realtime, so it is recommended to load all plugins to be used before playing.

## Examples

Here is an example of the dssinit opcode. It uses the file *dssi4cs.csd* [examples/dssi4cs.csd].

### Exemple 137. Example of the dssiinit opcode. (Remember to change the Library name)

```
<CsoundSynthesizer>
<CsOptions>
;use appropriate realtime options
</CsOptions>
<CsInstruments>
ksmps = 256
nchnls = 2
```

```

dssiinit

gihandle dssiinit "amp.so", 0, 1
;gihandle dssiinit "cmt.so", 30, 2
;gihandle2 dssiinit "cmt.so", 8, 1
;gihandle dssiinit "delayorama_1402", 0
gihandle2 dssiinit "cmt.so", 49, 1
;gihandle dssiinit "freq_tracker_1418.so", 0, 1, 1
;gihandle dssiinit "g2reverb.so", 0, 1
;gihandle2 dssiinit "declip_1195.so", 0, 1
;gihandle2 dssiinit "revdelay_1605.so", 0, 1
;gihandle2 dssiinit "tap_chorusflanger.so", 0, 1
;gihandle2 dssiinit "plate_1423.so", 0, 1
gihandle3 dssiinit "gate_1410.so", 0, 1
;gihandle3 dssiinit "hexter.so", 0, 1

instr 1
print p4
dssiactivate gihandle, p4
dssiactivate gihandle2, p4
dssiactivate gihandle3, p4
endin

instr 2
ain1 inch 1
ain2 inch 2
;aout1,aout2 dssiaudio gihandle, ain1, ain2
aout1 dssiaudio gihandle, ain1
outs aout1,aout1
endin

instr 3
kval linen 1, p3 /3, p3, p3/ 3
dssiactls gihandle, p4, kval, 1
endin

instr 4
ain1 inch 1
aout1 dssiaudio gihandle2, ain1
outs aout1,aout1
endin

</CsInstruments>
<CsScore>

i 1 1 1 1 1

i 2 2 15 ;plugin 1

i 3 3 12 0 ;Control port 0

i 4 8 2 ;plugin 2
e
</CsScore>
</CsoundSynthesizer>

```

## Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

# dssilist

dssilist — Lists all available DSSI and LADSPA plugins.

## Syntax

`dssilist`

## Description

*dssilist* checks the variables DSSI\_PATH and LADSPA\_PATH and lists all plugins available in all plugin libraries there.

LADSPA and DSSI libraries may contain more than one plugin which must be referenced by the index provided by *dssilist*.

This opcode produces a long printout which may interrupt realtime audio output, so it should be run at the start of a performance.

## Credits

2005

By: Andrés Cabrera

Uses code from Richard Furse's LADSPA sdk.

# dumpk

dumpk — Ecrit périodiquement la valeur d'un signal de contrôle de l'orchestre dans un fichier externe.

## Description

Ecrit périodiquement la valeur d'un signal de contrôle de l'orchestre dans un fichier externe, dans un format spécifique.

## Syntaxe

```
dumpk ksig, ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Peut être un nom de chemin complet avec un répertoire cible ou un simple nom de fichier à créer dans le répertoire courant.

*iformat* -- spécifie le format des données de sortie :

- 1 = caractères signés sur 8 bit (les 8 bit d'ordre supérieur d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII
- 8 = flottants en ASCII (2 positions décimales)

Noter que les sorties A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier de sortie ne contient pas d'information d'en-tête.

*iprd* -- la période de la sortie *ksig* en secondes, arrondie à la période de contrôle de l'orchestre la plus proche. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui créera un fichier de sortie échantillonné au taux de contrôle de l'orchestre.

## Exécution

*ksig* -- un signal au taux de contrôle

Cet opcode permet de sauvegarder la valeur d'un signal généré au taux de contrôle dans un fichier externe. Le fichier ne contient pas d'information auto-descriptive en en-tête. Mais il contient une suite temporelle échantillonnée régulièrement, appropriée pour une entrée ultérieure ou une analyse. Il peut y avoir n'importe quel nombre d'opcodes *dumpk* dans un instrument ou dans un orchestre mais chacun doit écrire dans un fichier différent.

## Exemples



Voici un exemple de l'opcode `dumpk`. Il utilise le fichier `dumpk.csd` [examples/dumpk.csd].

### Exemple 138. Exemple de l'opcode `dumpk`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dumpk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 20
nchnls = 1

; By Andres Cabrera 2008

instr 1
; Write fibonacci numbers to file "fibonacci.txt"
; as ascii long integers (mode 7), using the orchestra's
; control rate (iprd = 0)

knumber init 0
koldnumber init 1
ktrans init 1
ktrans = knumber
knumber = knumber + koldnumber
koldnumber = ktrans
dumpk knumber, "fibonacci.txt", 7, 0
printk2 knumber
endin

</CsInstruments>
<CsScore>

;Write to the file for 1 second. Since control rate is 20, 20 values will be written
i 1 0 1

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

`dumpk2`, `dumpk3`, `dumpk4`, `readk`, `readk2`, `readk3`, `readk4`

## Crédits

Par : John ffitch et Barry Vercoe

1999 ou avant

# dumpk2

`dumpk2` — Ecrit périodiquement les valeurs de deux signaux de contrôle de l'orchestre dans un fichier externe.

## Description

Ecrit périodiquement les valeurs de deux signaux de contrôle de l'orchestre dans un fichier externe, dans un format spécifique.

## Syntaxe

```
dumpk2 ksig1, ksig2, ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Peut être un nom de chemin complet avec un répertoire cible ou un simple nom de fichier à créer dans le répertoire courant.

*iformat* -- spécifie le format des données de sortie :

- 1 = caractères signés sur 8 bit (les 8 bit d'ordre supérieur d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII
- 8 = flottants en ASCII (2 positions décimales)

Noter que les sorties A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier de sortie ne contient pas d'information d'en-tête.

*iprd* -- la période de la sortie *ksig* en secondes, arrondie à la période de contrôle de l'orchestre la plus proche. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui créera un fichier de sortie échantillonné au taux de contrôle de l'orchestre.

## Exécution

*ksig1*, *ksig2* -- signaux au taux de contrôle.

Cet opcode permet de sauvegarder les valeurs de deux signaux générés au taux de contrôle dans un fichier externe. Le fichier ne contient pas d'information auto-descriptive en en-tête. Mais il contient une suite temporelle échantillonnée régulièrement, appropriée pour une entrée ultérieure ou une analyse. Il peut y avoir n'importe quel nombre d'opcodes *dumpk2* dans un instrument ou dans un orchestre mais chacun doit écrire dans un fichier différent.

## Exemples

Voir l'exemple de *dumpk*. La seule différence entre *dumpk* et *dumpk2* est que *dumpk2* peut écrire deux valeurs à la fois dans le fichier.

## Voir Aussi

*dumpk*, *dumpk3*, *dumpk4*, *readk*, *readk2*, *readk3*, *readk4*

## Crédits

Par : John ffitch et Barry Vercoe

1999 ou avant

# dumpk3

`dumpk3` — Ecrit périodiquement les valeurs de trois signaux de contrôle de l'orchestre dans un fichier externe.

## Description

Ecrit périodiquement les valeurs de trois signaux de contrôle de l'orchestre dans un fichier externe, dans un format spécifique.

## Syntaxe

```
dumpk3 ksig1, ksig2, ksig3, ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Peut être un nom de chemin complet avec un répertoire cible ou un simple nom de fichier à créer dans le répertoire courant.

*iformat* -- spécifie le format des données de sortie :

- 1 = caractères signés sur 8 bit (les 8 bit d'ordre supérieur d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII
- 8 = flottants en ASCII (2 positions décimales)

Noter que les sorties A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier de sortie ne contient pas d'information d'en-tête.

*iprd* -- la période de la sortie *ksig* en secondes, arrondie à la période de contrôle de l'orchestre la plus proche. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui créera un fichier de sortie échantillonné au taux de contrôle de l'orchestre.

## Exécution

*ksig1*, *ksig2*, *ksig3* -- signaux au taux de contrôle

Cet opcode permet de sauvegarder les valeurs de trois signaux générés au taux de contrôle dans un fichier externe. Le fichier ne contient pas d'information auto-descriptive en en-tête. Mais il contient une suite temporelle échantillonnée régulièrement, appropriée pour une entrée ultérieure ou une analyse. Il peut y avoir n'importe quel nombre d'opcodes *dumpk3* dans un instrument ou dans un orchestre mais chacun doit écrire dans un fichier différent.

## Exemples

Voir l'exemple de *dumpk*. La seule différence entre *dumpk* et *dumpk3* est que *dumpk3* peut écrire trois valeurs à la fois dans le fichier.

## Voir Aussi

*dumpk*, *dumpk2*, *dumpk4*, *readk*, *readk2*, *readk3*, *readk4*

## Crédits

Par : John ffitch et Barry Vercoe

1999 ou avant

# dumpk4

dumpk4 — Ecrit périodiquement les valeurs de quatre signaux de contrôle de l'orchestre dans un fichier externe.

## Description

Ecrit périodiquement les valeurs de quatre signaux de contrôle de l'orchestre dans un fichier externe, dans un format spécifique.

## Syntaxe

```
dumpk4 ksig1, ksig2, ksig3, ksig4, ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Peut être un nom de chemin complet avec un répertoire cible ou un simple nom de fichier à créer dans le répertoire courant.

*iformat* -- spécifie le format des données de sortie :

- 1 = caractères signés sur 8 bit (les 8 bit d'ordre supérieur d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII
- 8 = flottants en ASCII (2 positions décimales)

Noter que les sorties A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier de sortie ne contient pas d'information d'en-tête.

*iprd* -- la période de la sortie *ksig* en secondes, arrondie à la période de contrôle de l'orchestre la plus proche. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui créera un fichier de sortie échantillonné au taux de contrôle de l'orchestre.

## Exécution

*ksig1, ksig2, ksig3, ksig4* -- signaux au taux de contrôle

Cet opcode permet de sauvegarder les valeurs de quatre signaux générés au taux de contrôle dans un fichier externe. Le fichier ne contient pas d'information auto-descriptive en en-tête. Mais il contient une suite temporelle échantillonnée régulièrement, appropriée pour une entrée ultérieure ou une analyse. Il peut y avoir n'importe quel nombre d'opcodes *dumpk4* dans un instrument ou dans un orchestre mais chacun doit écrire dans un fichier différent.

## Exemples

Voir l'exemple de *dumpk*. La seule différence entre *dumpk* et *dumpk4* est que *dumpk4* peut écrire quatre valeurs à la fois dans le fichier.

## Voir Aussi

*dumpk*, *dumpk2*, *dumpk3*, *readk*, *readk2*, *readk3*, *readk4*

## Crédits

Par : John ffitch et Barry Vercoe

1999 ou avant

# duserrnd

duserrnd — Générateur de nombres aléatoires de distribution discrète définie par l'utilisateur.

## Description

Générateur de nombres aléatoires de distribution discrète définie par l'utilisateur.

## Syntaxe

```
aout duserrnd ktableNum
```

```
iout duserrnd itableNum
```

```
kout duserrnd ktableNum
```

## Initialisation

*itableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

## Exécution

*ktableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

*duserrnd* (Discrete USER-defined-distribution RaNDom generator) génère des nombres aléatoires selon une distribution aléatoire discrète créée par l'utilisateur. L'utilisateur peut créer l'histogramme de la distribution discrète au moyen de GEN41. Afin de créer cette table, on doit définir une quantité arbitraire de couples de nombres, le premier nombre de chaque paire représentant une valeur et le second représentant sa probabilité (voir GEN41 pour plus de détails).

Lorsqu'on l'utilise comme une fonction, le taux de génération dépend du type du taux de la variable d'entrée *XtableNum*. Dans ce cas, on peut l'insérer dans n'importe quelle formule. Le numéro de table peut varier au taux-k, ce qui permet de changer l'histogramme de la distribution durant l'exécution d'une note. *duserrnd* est destiné à être utilisé pour la génération de musique algorithmique.

On peut aussi utiliser *duserrnd* pour générer des valeurs suivant un ensemble d'intervalles de probabilités au moyen de fonctions de distribution générées par GEN42 (voir GEN42 pour plus de détails). Dans ce cas, si l'on veut simuler des intervalles continus, la longueur de la table *XtableNum* doit être raisonnablement grande car *duserrnd* ne fait pas d'interpolation entre les éléments de la table.

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Voir Aussi



*cusernd, urd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16

## else

else — Exécute un bloc de code lorsqu'une condition "if...then" est fausse.

## Description

Exécute un bloc de code lorsqu'une condition "if...then" est fausse.

## Syntaxe

`else`

## Exécution

*else* est utilisé dans un bloc de code entre les opcodes "*if...then*" et *endif*. Il définit les instructions à exécuter lorsqu'une condition "if...then" est fausse. Il ne peut y avoir qu'une seule instruction *else* et celle-ci doit être la dernière instruction conditionnelle avant l'opcode *endif*.

## Exemples

Voir l'exemple de l'opcode *if*.

## Voir Aussi

*elseif*, *endif*, *goto*, *if*, *igoto*, *kgoto*, *tigoto*, *timeout*

## Crédits

Nouveau dans la version 4.21

# elseif

elseif — Définit une autre condition "if...then" lorsqu'une condition "if...then" est fausse.

## Description

Définit une autre condition "if...then" lorsqu'une condition "if...then" est fausse.

## Syntaxe

```
elseif xa R xb then
```

où *R* est un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exécution

*elseif* est utilisé dans un bloc de code entre les opcodes "*if...then*" et *endif*. Lorsqu'une condition "if...then" est fausse, cela définit une autre condition "if...then" à tester. On peut utiliser n'importe quel nombre d'instructions *elseif*.

## Exemples

Voir l'exemple de l'opcode *if*.

## Voir Aussi

*else*, *endif*, *goto*, *if*, *igoto*, *kgoto*, *tigoto*, *timeout*

## Crédits

Nouveau dans la version 4.21

# endif

endif — Termine un bloc de code qui commence par une instruction "if...then".

## Description

Termine un bloc de code qui commence par une instruction "*if...then*".

## Syntaxe

`endif`

## Exécution

Tout bloc de code commençant par une instruction "*if...then*" doit se terminer par une instruction *endif*.

## Exemples

Voir l'exemple de l'opcode *if*.

## Voir Aussi

*elseif*, *else*, *goto*, *if*, *igoto*, *kgoto*, *tigoto*, *timeout*

## Crédits

Nouveau dans la version 4.21

# endin

endin — Termine un bloc d'instrument.

## Description

Termine le bloc d'instrument courant.

## Syntax

endin

## Initialisation

Termine le bloc d'instrument courant.

On peut définir les instruments dans n'importe quel ordre (mais ils seront toujours initialisés et exécutés par ordre de numéro d'instrument ascendant). Les blocs d'instruments ne peuvent pas être imbriqués (un bloc ne peut pas en contenir un autre).



### Note

Il peut y avoir n'importe quel nombre de blocs d'instrument dans un orchestre.

## Exemples

Voici un exemple de l'opcode *endin*. Il utilise le fichier *endin.csd* [examples/endin.csd].

### Exemple 139. Exemple de l'opcode *endin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d           ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o endin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0
```

```
    al oscils iamp, icps, iphs
    out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*instr*

## Crédits

Exemple écrit par Kevin Conder.

# endop

endop — Termine un bloc d'opcode défini par l'utilisateur.

## Description

Termine un bloc d'opcode défini par l'utilisateur.

## Syntaxe

endop

## Exécution

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode nom, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps ikmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

Le nouvel opcode peut ensuite être utilisé avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN] nom [xoutarg1] [, xoutarg2] ... [xoutargN] [, ikmps]
```

## Exemples

Voir l'exemple pour *opcode*.

## Voir Aussi

*opcode*, *setksmps*, *xin*, *xout*

## Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code de Matt J. Ingalls

Nouveau dans la version 4.22

# envlpx

envlpx — Applique une enveloppe constituée de 3 segments.

## Description

*envlpx* -- applique une enveloppe constituée de 3 segments :

1. une attaque dont la forme est donnée par une fonction
2. un pseudo entretien modifié exponentiellement
3. une chute exponentielle

## Syntaxe

```
ares envlpx xamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]
```

```
kres envlpx kamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]
```

## Initialisation

*irise* -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

*idur* -- durée globale en seconde. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

*idec* -- durée de la chute en secondes. Zéro signifie pas de chute. Si *idec* > *idur* la chute sera tronquée.

*ifn* -- numéro de la table de fonction avec point de garde dans laquelle la forme de l'attaque est stockée.

*iatss* -- facteur d'atténuation par lequel la dernière valeur de l'attaque d'*envlpx* évolue pendant le pseudo entretien de la note. Un facteur supérieur à 1 provoque une montée exponentielle tandis qu'un facteur inférieur à 1 crée une descente exponentielle. Un facteur égal à 1 maintient un véritable entretien de la note sur la dernière valeur de l'attaque. Il faut noter que cette atténuation n'évolue pas à vitesse constante (comme dans le cas du piano), mais qu'elle dépend de la durée de la note. Cependant, si *iatss* est négatif (ou si l'entretien < 4 périodes-k) une vitesse d'atténuation de *abs(iatss)* par seconde sera utilisée. 0 est interdit.

*iatdec* -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulles ou négatives sont interdites.

*ixmod* (facultatif, entre +- 0,9 environ) -- facteur de modification de courbe exponentielle, qui influe sur la raideur de la trajectoire exponentielle pendant l'entretien. Les valeurs négatives provoqueront une montée ou une descente accélérée (par exemple *subito piano*). Les valeurs positives provoqueront une montée ou une descente ralentie. La valeur par défaut est zéro (exponentielle non modifiée).

## Exécution

*kamp*, *xamp* -- amplitude du signal d'entrée.



Les modifications de l'attaque sont appliquées pendant les premières *irise* secondes, et celles de la chute à partir de *idur* - *idec*. Si ces périodes sont séparées dans le temps il y aura un entretien au cours duquel *amp* sera modifié selon le schéma exponentiel décrit. Si l'attaque et la chute se chevauchent alors les deux modifications agiront simultanément durant cette période commune. Si la durée globale *idur* est dépassée pendant l'exécution, la chute continuera dans la même direction, en tendant asymptotiquement vers zéro.

## Exemples

Voici un exemple de l'opcode *envlpx*. Il utilise le fichier *envlpx.csd* [examples/envlpx.csd].

### Exemple 140. Exemple de l'opcode *envlpx*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o envlpx.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a simple instrument.
instr 1
; Set the amplitude.
kamp init 20000
; Get the frequency from the fourth p-field.
kcps = cpspch(p4)

al vco kamp, kcps, 1
out al
endin

; Instrument #2 - instrument with an amplitude envelope.
instr 2
kamp = 20000
irise = 0.05
idur = p3 - .01
idec = 0.5
ifn = 2
iatss = 1
iatdec = 0.01

; Create an amplitude envelope.
kenv envlpx kamp, irise, idur, idec, ifn, iatss, iatdec

; Get the frequency from the fourth p-field.
kcps = cpspch(p4)

al vco kenv, kcps, 1
out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1
; Table #2, a rising envelope.
```

```
f 2 0 129 -7 0 128 1

; Set the tempo to 120 beats per minute.
t 0 120

; Make sure the score plays for 33 seconds.
f 0 33

; Play a melody with Instrument #1.
; p4 = frequency in pitch-class notation.
i 1 0 1 8.04
i 1 1 1 8.04
i 1 2 1 8.05
i 1 3 1 8.07
i 1 4 1 8.07
i 1 5 1 8.05
i 1 6 1 8.04
i 1 7 1 8.02
i 1 8 1 8.00
i 1 9 1 8.00
i 1 10 1 8.02
i 1 11 1 8.04
i 1 12 2 8.04
i 1 14 2 8.02

; Repeat the melody with Instrument #2.
; p4 = frequency in pitch-class notation.
i 2 16 1 8.04
i 2 17 1 8.04
i 2 18 1 8.05
i 2 19 1 8.07
i 2 20 1 8.07
i 2 21 1 8.05
i 2 22 1 8.04
i 2 23 1 8.02
i 2 24 1 8.00
i 2 25 1 8.00
i 2 26 1 8.02
i 2 27 1 8.04
i 2 28 2 8.04
i 2 30 2 8.02
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*envlpxr, linen, linenr*

## Crédits

Merci à Luis Jure pour avoir signalé une erreur avec *iatss*.

Exemple écrit par Kevin Conder.

# envlpxr

envlpxr — L'opcode *envlpx* avec un segment final de relâchement.

## Description

*envlpxr* est le même que *envlpx* sauf que le segment final n'est exécuté qu'après un évènement MIDI de relâchement de note. La note est ensuite allongée de la durée de la chute.

## Syntaxe

```
ares envlpxr xamp, irise, idec, ifn, iatss, iatdec [, ixmod] [, irind]
```

```
kres envlpxr kamp, irise, idec, ifn, iatss, iatdec [, ixmod] [, irind]
```

## Initialisation

*irise* -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

*idec* -- durée de la chute en secondes. Zéro signifie pas de chute.

*ifn* -- numéro de la table de fonction avec point de garde dans laquelle la forme de l'attaque est stockée.

*iatss* -- facteur d'atténuation par lequel la dernière valeur de l'attaque d'*envlpxr* évolue pendant le pseudo entretien de la note. Un facteur supérieur à 1 provoque une montée exponentielle tandis qu'un facteur inférieur à 1 crée une descente exponentielle. Un facteur égal à 1 maintient un véritable entretien de la note sur la dernière valeur de l'attaque. Il faut noter que cette atténuation n'évolue pas à vitesse constante (comme dans le cas du piano), mais qu'elle dépend de la durée de la note. Cependant, si *iatss* est négatif (ou si l'entretien < 4 périodes-k) une vitesse d'atténuation de *abs(iatss)* par seconde sera utilisée. 0 est interdit.

*iatdec* -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulles ou négatives sont interdites.

*ixmod* (facultatif, entre +- 0,9 environ) -- facteur de modification de courbe exponentielle, qui influe sur la raideur de la trajectoire exponentielle pendant l'entretien. Les valeurs négatives provoqueront une montée ou une descente accélérée (par exemple *subito piano*). Les valeurs positives provoqueront une montée ou une descente ralentie. La valeur par défaut est zéro (exponentielle non modifiée).

*irind* (facultatif) -- indicateur d'indépendance. S'il est nul, la durée de relâchement (*idec*) aura une influence sur l'allongement de la note après un note-off. S'il est non nul, la durée *idec* sera relativement indépendante de l'allongement de la note (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*kamp*, *xamp* -- amplitude du signal d'entrée.

*envlpxr* fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un évènement ou MIDI note-off est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *idec* secondes à moins qu'il ne soit rendu indépendant par *irind*. Dans ce cas, la chute démarrera de l'endroit, quel qu'il soit, où l'on se trou-

vait à ce moment précis.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note-off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *envlpxr*, car la durée est allongée automatiquement.

Ces unités « r » peuvent être modifiées également par des événements MIDI note-off provoqués par une vitesse nulle. Si l'indicateur *irind* est positionné (différent de zéro), la durée d'exécution totale n'est pas affectée par les données de note-off ou de vitesse nulle.

**Unités « r » multiples.** Quand plusieurs unités « r » sont présentes dans le même instrument, il est habituel qu'une seule d'entre elle influence la durée totale de la note. C'est normalement l'unité contrôlant l'amplitude principale de la note. D'autres unités contrôlant par exemple l'évolution d'un filtre, peuvent toujours être sensibles aux commandes note-off tout en n'affectant pas la durée grâce à leur indépendance (*irind* non nul). En fonction de leur propre valeur *idec* (durée de relâchement), les unités « r » indépendantes pourront ou ne pourront pas atteindre leur destination finale avant que la note ne se termine. Si elles y arrivent, elles tiendront simplement leur dernière valeur jusqu'à la fin. Si plusieurs unités « r » sont principales, l'extension de la note sera celle de la plus grande valeur *idec*.

## Voir Aussi

*envlpx*, *linen*, *linenr*

## Crédits

Merci à Luis Jure pour avoir signalé une erreur avec *iatss*.

# ephasor

ephasor —

## Performance

Ephasor has been added to Csound 5.10, but its behavior will change for 5.11. Stay tuned...

## Credits

Author: Victor Lazzarini  
2008

New in version 5.10

# eqfil

eqfil — Equalizer filter

## Description

The opcode eqfil is a 2nd order tunable equalisation filter based on Regalia and Mitra design ("Tunable Digital Frequency Response Equalization Filters", IEEE Trans. on Ac., Sp. and Sig Proc., 35 (1), 1987). It provides a peak/notch filter for building parametric/graphic equalisers.

The amplitude response for this filter will be flat (=1) for kgain=1. With kgain is bigger than 1, there will be a peak at the centre frequency, whose width is given by the kbw parameter, but outside this band, the response will tend towards 1. Conversely, if kgain is smaller than 1, a notch will be created around the CF.

## Syntax

```
asig eqfil ain, kcf, kbw, kgain[, istor]
```

## Initialization

*istor* --initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*asig* -- filtered output signal.

*ain* -- input signal.

*kcf* -- filter centre frequency

*kbw* -- peak/notch bandwidth (Hz).

*kgain* -- peak/notch gain.

## Examples

### Exemple 141. Example

```
kfe      expseg 10, p3*0.9, 180, p3*0.1, 175
kenv     linen 1000, 0.05, p3, 0.05
asig     buzz  kenv, kfe, sr/(2*kfe), 1
afil     eqfil asig, 1500, 400, 0.1

out afil
```

## Credits

Author: Victor Lazzarini  
April 2007

New in version 5.06

# event

event — Génère un évènement de partition à partir d'un instrument.

## Description

Génère un évènement de partition à partir d'un instrument.

## Syntaxe

```
event "scorechar", kinsnum, kdelay, kdur, [, kp4] [, kp5] [, ...]
```

```
event "scorechar", "insname", kdelay, kdur, [, kp4] [, kp5] [, ...]
```

## Initialisation

« *scorechar* » -- Une chaîne de caractères (entre guillemets) représentant le p-champ initial dans une instruction de partition. C'est habituellement « *e* », « *f* », ou « *i* ».

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

## Exécution

*kinsnum* -- L'instrument à utiliser pour l'évènement. Cela correspond au premier p-champ, p1, dans une instruction de partition.

*kdelay* -- Quand (en secondes) l'évènement aura lieu à partir de l'instant courant de l'exécution. Cela correspond au second p-champ, p2, dans une instruction de partition.

*kdur* -- Durée (en secondes) de l'évènement. Cela correspond au troisième p-champ, p3, dans une instruction de partition.

*kp4*, *kp5*, ... (facultatif) -- Paramètres représentant des p-champs supplémentaires dans une instruction de partition. Ils commencent par le quatrième p-champ, p4.



### Note

Noter que l'opcode *event* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici un exemple de l'opcode event. Il utilise le fichier *event.csd* [examples/event.csd].

### Exemple 142. Exemple de l'opcode event.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o event.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - an oscillator with a high note.
instr 1
; Create a trigger and set its initial value to 1.
ktrigger init 1

; If the trigger is equal to 0, continue playing.
; If not, schedule another event.
if (ktrigger == 0) goto contin
; kscoreop="i", an i-statement.
; kinsnum=2, play Instrument #2.
; kwhen=1, start at 1 second.
; kdur=0.5, play for a half-second.
event "i", 2, 1, 0.5

; Make sure the event isn't triggered again.
ktrigger = 0

contin:
al oscils 10000, 440, 1
out al
endin

; Instrument #2 - an oscillator with a low note.
instr 2
al oscils 10000, 220, 1
out al
endin

</CsInstruments>
<CsScore>

; Make sure the score plays for two seconds.
f 0 2

; Play Instrument #1 for a half-second.
i 1 0 0.5
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode event utilisant un instrument nommé. Il utilise le fichier *event\_named.csd* [examples/event\_named.csd].

### Exemple 143. Exemple de l'opcode event utilisant un instrument nommé.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o event_named.wav -W ;; for file output any platform
</CsOptions>

```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - an oscillator with a high note.
instr 1
; Create a trigger and set its initial value to 1.
ktrigger init 1

; If the trigger is equal to 0, continue playing.
; If not, schedule another event.
if (ktrigger == 0) goto contin
; kscoreop="i", an i-statement.
; kinsnum="low_note", instrument named "low_note".
; kwhen=1, start at 1 second.
; kdur=0.5, play for a half-second.
event "i", "low_note", 1, 0.5

; Make sure the event isn't triggered again.
ktrigger = 0

contin:
al oscils 10000, 440, 1
out al
endin

; Instrument "low_note" - an oscillator with a low note.
instr low_note
al oscils 10000, 220, 1
out al
endin

</CsInstruments>
<CsScore>

; Make sure the score plays for two seconds.
f 0 2

; Play Instrument #1 for a half-second.
i 1 0 0.5
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*event\_i, schedule, schedwhen, schedkwhen, schedkwhennamed, scoreline, scoreline\_i*

## Crédits

Exemples écrits par Kevin Conder.

Nouveau dans la version 4.17

Merci à Matt Ingalls pour son aide à corriger l'exemple.

Merci à Matt Ingalls pour son aide à clarifier le paramètre kwhen/kdelay.

# event\_i

event\_i — Génère un évènement de partition à partir d'un instrument.

## Description

Génère un évènement de partition à partir d'un instrument.

## Syntaxe

```
event_i "scorechar", iinsnum, idelay, idur, [, ip4] [, ip5] [, ...]
```

```
event_i "scorechar", "insname", idelay, idur, [, ip4] [, ip5] [, ...]
```

## Initialisation

« *scorechar* » -- Une chaîne de caractères (entre guillemets) représentant le p-champ initial d'une instruction de partition. C'est habituellement « *e* », « *f* », ou « *i* ».

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*iinsnum* -- L'instrument à utiliser pour cet évènement. Cela correspond au premier p-champ, p1, dans une instruction de partition.

*idelay* -- Quand (en secondes) l'évènement aura lieu à partir de l'instant courant de l'exécution. Cela correspond au second p-champ, p2, dans une instruction de partition.

*idur* -- Durée (en secondes) de l'évènement. Cela correspond au troisième p-champ, p3, dans une instruction de partition.

*ip4*, *ip5*, ... (facultatif) -- Paramètres représentant des p-champs supplémentaires dans une instruction de partition. Ils commencent par le quatrième p-champ, p4.

## Exécution

L'évènement est ajouté à la file d'attente pendant la phase d'initialisation.



### Note

Noter que l'opcode *event\_i* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Voir Aussi

*event*, *schedule*, *schedwhen*, *schedkwhen*, *schedkwhennamed*, *scoreline*, *scoreline\_i*

## Crédits

Ecrit par Istvan Varga.

Nouveau dans Csound5

# exitnow

exitnow — Quitte Csound aussi vite que possible, sans nettoyage.

## Description

Dans Csound4 cela appelle une fonction de sortie pour quitter Csound aussi vite que possible. Dans Csound5, on revient au code appelant.

## Syntaxe

`exitnow`

## Exécution

Arrête Csound lors du cycle d'initialisation.

# exp

exp — Retourne  $e$  élevé à la puissance  $x$ .

## Description

Retourne  $e$  élevé à la puissance  $x$ .

## Syntaxe

**exp**( $x$ ) (pas de restriction de taux)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode exp. Il utilise le fichier *exp.csd* [examples/exp.csd].

### Exemple 144. Exemple de l'opcode exp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o exp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = exp(8)
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 2980.958
```

## Voir Aussi

*abs, frac, int, log, log10, i, sqrt*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.21

# expcurve

expcurve — Cet opcode implémente une formule qui génère une courbe exponentielle normalisée dans l'intervalle 0 - 1. Il est basé sur le travail dans Max / MSP de Eric Singer (c) 1994.

## Description

Génère une courbe exponentielle dans l'intervalle de 0 à 1 avec une raideur de pente arbitraire. Une raideur de pente inférieure ou égale à 1,0 lévera des erreurs NaN (Not-a-Number) et provoquera un comportement instable.

La formule utilisée pour le calcul de la courbe est :

$$(\exp(x * \log(y)) - 1) / (y - 1)$$

où x est égal à *kindex* et y est égal à *ksteepness*.

## Syntaxe

kout **expcurve** kindex, ksteepness

## Exécution

*kindex* -- Valeur d'indice. Attendue dans l'intervalle de 0 à 1.

*ksteepness* -- Raideur de la courbe générée. Avec des valeurs proches de 1,0 on obtient une courbe plus rectiligne alors qu'avec des valeurs plus grandes la courbe est plus raide.

*kout* -- Sortie pondérée.

## Exemples

Voici un exemple de l'opcode expcurve. Il utilise le fichier *expcurve.csd* [examples/expcurve.csd].

### Exemple 145. Exemple de l'opcode expcurve.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent
-odac        -iadc     -d      ;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 1000
nchnls = 2

instr 1 ; logcurve test

kmod phasor 1/p3
kout expcurve kmod, p4
```



```
printks "mod = %f  out= %f\\n", 0.5, kmod, kout
      endin

/*--- ---*/
</CsInstruments>
<CsScore>

i1 0 5 2
i1 5 5 5
i1 10 5 30
i1 15 5 0.5

e
</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*scale, gainslider, logcurve*

## Crédits

Auteur : David Akbari  
Octobre  
2006

# expon

expon — Trace une courbe exponentielle entre les points spécifiés.

## Description

Trace une courbe exponentielle entre les points spécifiés.

## Syntaxe

```
ares expon ia, idur, ib
```

```
kres expon ia, idur, ib
```

## Initialisation

*ia* -- valeur initiale. Zéro est interdit pour les exponentielles.

*ib* -- valeur après *idur* secondes. Pour les exponentielles, doit être non nulle et du même signe que *ia*.

*idur* -- durée en secondes du segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par deux points spécifiés. La valeur de *idur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le segment continuera dans la même direction.

## Exemples

Voici un exemple de l'opcode *expon*. Il utilise le fichier *expon.csd* [examples/expon.csd].

### Exemple 146. Exemple de l'opcode *expon*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expon.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
; Define kcps as a frequency value that exponentially declines
; from 880 to 220. It declines over the period set by p3.
kcps expon 880, p3, 220

a1 oscil 20000, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*expseg, expsegr, line, linseg, linsegr*

## Crédits

Exemple écrit par Kevin Conder.

# exprand

exprand — Générateur de nombres aléatoires de distribution exponentielle (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution exponentielle (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares exprand klambda
```

```
ires exprand klambda
```

```
kres exprand klambda
```

## Exécution

*klambda* -- paramètre lambda pour la distribution exponentielle.

La fonction de densité de probabilité d'une distribution exponentielle est une courbe exponentielle, dont la moyenne est  $0,69515/\lambda$ . Pour des explications plus détaillées de ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode exprand. Il utilise le fichier *exprand.csd* [examples/exprand.csd].

### Exemple 147. Exemple de l'opcode exprand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; -o exprand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  ; Generate a random between 0 and 1.
  ; krange = 1

  i1 exprand 1

  print i1
endin

; Instrument #2.
instr 2
  ; Generate a random between 0 and 1.
  ; krange = 1

  seed 0

  i1 exprand 1

  print i1
endin

</CsInstruments>
<CsScore>
; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 0.174
```

## Voir Aussi

*seed, betarand, bexprnd, cauchy, gauss, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur: Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.

# expseg

expseg — Trace une suite de segments d'exponentielle entre les points spécifiés.

## Description

Trace une suite de segments d'exponentielle entre les points spécifiés.

## Syntaxe

```
ares expseg ia, idur1, ib [, idur2] [, ic] [...]
```

```
kres expseg ia, idur1, ib [, idur2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale. Zéro est interdit pour les exponentielles.

*ib, ic*, etc. -- valeur après *dur1* secondes, etc. Pour les exponentielles, doivent être différentes de zéro et du même signe que *ia*.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2, idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

Noter que l'opcode *expseg* n'opère pas correctement au taux audio lorsque les segments sont plus courts qu'une k-période. Dans ce cas, il vaut mieux utiliser l'opcode *expsega*.

## Exemples

Voici un exemple de l'opcode *expseg*. Il utilise le fichier *expseg.csd* [examples/expseg.csd].

### Exemple 148. Exemple de l'opcode expseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in
```

```

-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; p4 = frequency in pitch-class notation.
kcps = cpspch(p4)

; Create an amplitude envelope.
kenv expseg 0.01, p3*0.25, 1, p3*0.75, 0.01
kamp = kenv * 30000

al oscil kamp, kcps, 1
out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*expon, expsega, expsegr, line, linseg, linsegr transeg*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans Csound 3.57

# expsega

expsega — Un générateur de segments exponentiels opérant au taux-a.

## Description

Un générateur de segments exponentiels opérant au taux-a. Cette unité est pratiquement identique à *expseg*, mais elle est plus précise lorsque l'on définit des segments de courte durée (c-à-d., dans une phase d'attaque percussive) au taux audio.

## Syntaxe

```
ares expsega ia, idur1, ib [, idur2] [, ic] [...]
```

## Initialiation

*ia* -- valeur initiale. Zéro est interdit.

*ib*, *ic*, etc. -- valeur après *idur1* secondes, etc. Doivent être non nulles et de même signe que *ia*.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

## Exécution

Cette unité génère des signaux audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

## Exemples

Voici une exemple de l'opcode expsega. Il utilise le fichier *expsega.csd* [examples/expsega.csd].

### Exemple 149. Exemple de l'opcode expsega.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expsega.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```



```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Define a short percussive amplitude envelope that
; goes from 0.01 to 20,000 and back.
aenv expsega 0.01, 0.1, 20000, 0.1, 0.01

al oscil aenv, 440, 1
out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #1 for one second.
i 1 1 1
; Play Instrument #1 for one second.
i 1 2 1
; Play Instrument #1 for one second.
i 1 3 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*expseg, expsegr*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans Csound 3.57

## expsegr

`expsegr` — Trace une suite de segments d'exponentielle entre les points spécifiés avec un segment de relâchement.

## Description

Trace une suite de segments d'exponentielle entre les points spécifiés avec un segment de relâchement (fin de l'entretien de la note).

## Syntaxe

```
ares expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

```
kres expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

## Initialisation

*ia* -- valeur initiale. Zéro est interdit pour les exponentielles.

*ib*, *ic*, etc. -- valeur après *dur1* secondes, etc. Pour les exponentielles, doivent être différentes de zéro et du même signe que *ia*.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

*irel*, *iz* -- durée en secondes et valeur finale du segment de relâchement de la note.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

`expsegr` fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un événement ou MIDI noteoff est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *irel* secondes, de façon à ce que la valeur *iz* soit atteinte à la fin de cette période (quelque soit le segment dans lequel se trouvait l'unité). Les unités « r » peuvent aussi être modifiées par les vélocités nulles provoquant un message MIDI noteoff. S'il y a plusieurs extensions de durée dans un instrument, c'est la plus longue qui sera choisie.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme `linsegr` et `madsr`, ou bien l'on peut construire des enveloppes plus complexes au moyen de `xtratim` et de `release`. Noter que qu'il n'est pas nécessaire d'utiliser `xtratim` avec `expsegr`, car la durée est allongée automatiquement.

## Exemples

Voici un exemple de l'opcode `expsegr`. Il utilise le fichier `expsegr.csd` [examples/expsegr.csd].

### Exemple 150. Exemple de l'opcode `expsegr`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o expsegr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; p4 = frequency in pitch-class notation.
kcps = cpspch(p4)

; Use an amplitude envelope with second-long release.
kenv expsegr 0.01, p3/2, 1, p3/2, 0.01, 1, 1
kamp = kenv * 30000

a1 oscil kamp, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Make sure the score lasts for four seconds.
f 0 4

; p4 = frequency (in pitch-class notation).
; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*linsegr, expsegr, envlpxr, mxadsr, madsr expon, expseg, expsega, xtratim*

## Crédits

Auteur : Barry L. Vercoe

Exemple écrit par Kevin Conder.

Nouveau dans Csound 3.47

# ficlose

ficlose — Ferme un fichier ouvert précédemment.

## Description

*ficlose* peut être utilisé pour fermer un fichier qui avait été ouvert avec *fiopen*.

## Syntaxe

```
ficlose ihandle
```

```
ficlose Sfilename
```

## Initialisation

*ihandle* -- un nombre qui identifie le fichier (généré par un *fiopen* précédent).

*Sfilename* -- une chaîne de caractères entre guillemets ou une variable chaîne de caractères contenant le nom du fichier. Il faut donner le chemin complet si le répertoire du fichier n'est pas dans le PATH (chemin) du système et s'il n'est pas dans le répertoire courant.

## Exécution

*ficlose* ferme un fichier ouvert précédemment avec *fiopen*. *ficlose* n'est nécessaire que si l'on désire lire un fichier écrit durant la même exécution de Csound, car Csound ne sauve les données dans tous les fichiers ouverts et ne ferme ceux-ci que lorsqu'il termine une exécution. L'opcode *ficlose* est utile par exemple si l'on veut sauvegarder des presets dans des fichiers auxquels on veut pouvoir accéder sans terminer Csound.



### Note

Si l'on a pas besoin de cette fonctionnalité, il est plus sûr de ne pas appeler *ficlose*, et de laisser Csound fermer les fichiers lorsqu'il se termine.

Si un fichier fermé par *ficlose* est accédé par un autre opcode (comme *fout* ou *foutk*), il sera fermé plus tard lorsqu'il ne sera plus utilisé.



### Avertissement

Il faut utiliser cet opcode avec précaution, car l'identificateur de fichier n'est plus valide, et il y aura une erreur d'initialisation si un opcode essaie d'accéder au fichier fermé.

## Voir Aussi

*fout*, *fout*, *fouti*, *foutir*, *foutk*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 5.02 de Csound

# filebit

filebit — Returns the number of bits in each sample in a sound file.

## Description

Returns the number of bits in each sample in a sound file.

## Syntax

```
ir filebit ifilcod [, iallowraw]
```

## Initialization

*ifilcod* -- sound file to be queried

*iallowraw* -- (Optional) Allow raw sound files (default=1)

## Performance

*filebit* returns the number of bits in each sample in the sound file *ifilcod*. In the case of floating point samples the value -1 is returned for floats and -2 for doubles. For non-PCM formats the value is negative, and based on libsndfile's format encoding.

## Examples

Here is an example of the filebit opcode. It uses the file *filebit.csd* [examples/filebit.csd], and *mary.wav* [examples/mary.wav].

### Exemple 151. Example of the filebit opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac        -iadc    ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o filebit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the number of channels in the
; audio file "mary.wav".
ibits filebit "mary.wav"
print ibits
```

```
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

The audio file « mary.wav » is in monoaural CD format, so *filebit*'s output should include a line like this:

```
instr 1:  ibits = 8.000
```

## See Also

*filelen*, *filenchnls*, *filepeak*, *filesr*

## Credits

Author: Victor Lazzarini  
July 1999

Example written by John ffitich.

New in Csound version 5.11



# filelen

filelen — Retourne la longueur d'un fichier son.

## Description

Retourne la longueur d'un fichier son.

## Syntaxe

```
ir filelen ifilcod, [iallowraw]
```

## Initialisation

*ifilcod* -- fichier son à interroger

*iallowraw* -- (facultatif) permet des fichiers son bruts (vaut 1 par défaut)

## Exécution

*filelen* retourne la longueur du fichier son *ifilcod* en secondes. *filelen* peut retourner la longueur des fichiers de type convolve et PVOC si le paramètre *iallowraw* est différent de zéro (il est non nul par défaut).

## Exemples

Voici un exemple de l'opcode *filelen*. Il utilise les fichiers *filelen.csd* [examples/filelen.csd] et *mary.wav* [examples/mary.wav].

### Exemple 152. Exemple de l'opcode *filelen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o filelen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the length of the audio file
; "mary.wav" in seconds.
ilen filelen "mary.wav"
print ilen
```

```
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Le fichier audio « mary.wav » dure 3.5 secondes. Ainsi la sortie de *filelen* contiendra une ligne comme :

```
instr 1:  ilen = 3.501
```

## Voir Aussi

*filebit, filenchnls, filepeak, filesr*

## Crédits

Auteur : Matt Ingalls  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# filenchnls

filenchnls — Retourne le nombre de canaux d'un fichier son.

## Description

Retourne le nombre de canaux d'un fichier son.

## Syntaxe

```
ir filenchnls ifilcod [, iallowraw]
```

## Initialisation

*ifilcod* -- fichier son à interroger

*iallowraw* -- (facultatif) permet des fichiers son bruts (vaut 1 par défaut)

## Exécution

*filenchnls* retourne le nombre de canaux du fichier son *ifilcod*. *filenchnls* peut retourner le nombre de canaux des fichiers de type convolve et PVOC si le paramètre *iallowraw* est différent de zéro (il est non nul par défaut).

## Exemples

Voici un exemple de l'opcode *filenchnls*. Il utilise les fichiers *filenchnls.csd* [examples/filenchnls.csd] et *mary.wav* [examples/mary.wav].

### Exemple 153. Exemple de l'opcode *filenchnls*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o filenchnls.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the number of channels in the
; audio file "mary.wav".
ichnls filenchnls "mary.wav"
print ichnls
```

```
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Le fichier audio « mary.wav » est monophonique (1 canal). Ainsi la sortie de *filenchnls* contiendra une ligne comme :

```
instr 1:  ichnls = 1.000
```

## Voir Aussi

*filebit, filelen, filepeak, filesr*

## Crédits

Auteur : Matt Ingalls  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# filepeak

filepeak — Retourne la valeur absolue de la crête d'un fichier son.

## Description

Retourne la valeur absolue de la crête d'un fichier son.

## Syntaxe

```
ir filepeak ifilcod [, ichnl]
```

## Initialisation

*ifilcod* -- fichier son à interroger

*ichnl* (facultatif, 0 par défaut) -- canal sur lequel la valeur de crête est calculée. La valeur par défaut est 0.

- *ichnl* = 0 retourne la valeur de crête de tous les canaux
- *ichnl* > 0 retourne la valeur de crête de *ichnl*

## Exécution

*filepeak* retourne la valeur absolue de la crête du fichier son *ifilcod*.

## Exemples

Voici un exemple de l'opcode filepeak. Il utilise les fichiers *filepeak.csd* [examples/filepeak.csd] et *mary.wav* [examples/mary.wav].

### Exemple 154. Exemple de l'opcode filepeak.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o filepeak.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
; Print out the peak absolute value of the
; audio file "mary.wav".
ipeak filepeak "mary.wav"
print ipeak
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

La valeur absolue de la crête du fichier son « mary.wav » est 0.306902. Ainsi la sortie de *filepeak* contiendra une ligne comme :

```
instr 1: ipeak = 0.307
```

## Voir Aussi

*filelen, filenchnls, filesr*

## Crédits

Auteur : Matt Ingalls  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# filesr

filesr — Retourne le taux d'échantillonnage d'un fichier son.

## Description

Retourne le taux d'échantillonnage d'un fichier son.

## Syntaxe

```
ir filesr ifilcod [, iallowraw]
```

## Initialisation

*ifilcod* -- fichier son à interroger

*iallowraw* -- (facultatif) permet des fichiers son bruts (vaut 1 par défaut)

## Exécution

*filesr* retourne le taux d'échantillonnage du fichier son *ifilcod*. *filesr* peut retourner le taux d'échantillonnage des fichiers de type convolve et PVOC si le paramètre *iallowraw* est différent de zéro (il est non nul par défaut).

## Exemples

Voici un exemple de l'opcode *filesr*. Il utilise les fichiers *filesr.csd* [examples/filesr.csd] et *mary.wav* [examples/mary.wav].

### Exemple 155. Exemple de l'opcode *filenchnls*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o filesr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the sampling rate of the
; audio file "mary.wav".
isr filesr "mary.wav"
print isr
```

```
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Le fichier audio « mary.wav » a été échantillonné à 44.1 KHz. Ainsi la sortie de *filesr* contiendra une ligne comme :

```
instr 1:  isr = 44100.000
```

## Voir Aussi

*filebit, filelen, filenchnls, filepeak*

## Crédits

Auteur : Matt Ingalls  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound



## filter2

**filter2** — Performs filtering using a transposed form-II digital filter lattice with no time-varying control.

## Description

General purpose custom filter with time-varying pole control. The filter coefficients implement the following difference equation:

$$(1)*y(n) = b0*x[n] + b1*x[n-1] + \dots + bM*x[n-M] - a1*y[n-1] - \dots - aN*y[n-N]$$

the system function for which is represented by:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + \dots + bM*Z^{-M}}{1 + a1*Z^{-1} + \dots + aN*Z^{-N}}$$

## Syntax

```
ares filter2 asig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
```

```
kres filter2 ksig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
```

## Initialization

At initialization the number of zeros and poles of the filter are specified along with the corresponding zero and pole coefficients. The coefficients must be obtained by an external filter-design application such as Matlab and specified directly or loaded into a table via *GEN01*.

## Performance

The *filter2* opcodes perform filtering using a transposed form-II digital filter lattice with no time-varying control.

Since *filter2* implements generalized recursive filters, it can be used to specify a large range of general DSP algorithms. For example, a digital waveguide can be implemented for musical instrument modeling using a pair of *delayr* and *delayw* opcodes in conjunction with the *filter2* opcode.

## Examples

A first-order linear-phase lowpass linear-phase FIR filter operating on a k-rate signal:

```
k1 filter2 ksig, 2, 0, 0.5, 0.5 ;; k-rate FIR filter
```

## See Also

*zfilter2*

## Credits

Author: Michael A. Casey  
M.I.T.  
Cambridge, Mass.  
1997

New in version 3.47

# fin

*fin* — Lit des signaux depuis un fichier au taux-a.

## Description

Lit des signaux depuis un fichier au taux-a.

## Syntaxe

```
fin ifilename, iskipframes, iformat, ain1 [, ain2] [, ain3] [...]
```

## Initialisation

*ifilename* -- nom du fichier d'entrée (peut être une chaîne de caractères ou un identificateur numérique généré par *fopen*).

*iskipframes* -- nombre de trames à ignorer au début (chaque trame contient un échantillon de chaque canal).

*iformat* -- un nombre spécifiant le format du fichier d'entrée pour les fichiers sans en-tête. Si un en-tête est trouvé, cet argument est ignoré.

- 0 - flottants sur 32 bit sans en-tête
- 1 - entiers sur 16 bit sans en-tête

## Exécution

*fin* (file input) est le complément de *fout* : il lit un fichier multicanaux pour générer des signaux de taux audio. Il faut s'assurer que le nombre de canaux du fichier d'entrée est le même que le nombre d'arguments *ainX*.



### Note

Prière de noter que comme cet opcode génère sa sortie en utilisant des paramètres d'entrée (placés à droite de l'opcode), ces variables doivent avoir été initialisées avant leur utilisation, sinon une erreur "utilisé avant d'être défini" se produira. On peut utiliser l'opcode *init* pour cela.

## Voir Aussi

*fini*, *fink*

## Crédits

Auteur : Gabriel Maldonado  
Italie

1999

Nouveau dans la version 3.56 de Csound

# fini

**fini** — Lit des signaux depuis un fichier au taux-i.

## Description

Lit des signaux depuis un fichier au taux-i.

## Syntaxe

```
fini ifilename, iskipframes, iformat, in1 [, in2] [, in3] [, ...]
```

## Initialisation

*ifilename* -- nom du fichier d'entrée (peut être une chaîne de caractères ou un identificateur numérique généré par *fiopen*).

*iskipframes* -- nombre de trames à ignorer au début (chaque trame contient un échantillon de chaque canal).

*iformat* -- un nombre spécifiant le format du fichier d'entrée pour les fichiers sans en-tête. Si un en-tête est trouvé, cet argument est ignoré.

- 0 - flottants en format texte (avec boucle ; voir ci-dessous)
- 1 - flottants en format texte (sans boucle ; voir ci-dessous)
- 2 - flottants sur 32 bit en format binaire (sans boucle)

## Exécution

*fini* est le complément de *fouti* et de *foutir*. Il lit les valeurs chaque fois qu'une note de l'instrument correspondant est activée. Lorsque *iformat* vaut 0 et que la fin du fichier est atteinte, le pointeur de fichier est remis à zéro. Ceci redémarre la lecture depuis le début. Lorsque *iformat* vaut 1 ou 2, aucune boucle n'est active et à la fin du fichier, les variables correspondantes sont remplies avec des zéros.



### Note

Prière de noter que comme cet opcode génère sa sortie en utilisant des paramètres d'entrée (placés à droite de l'opcode), ces variables doivent avoir été initialisées avant leur utilisation, sinon une erreur "utilisé avant d'être défini" se produira. On peut utiliser l'opcode *init* pour cela.

## Voir Aussi

*fin*, *fink*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# fink

fink — Lit des signaux depuis un fichier au taux-k.

## Description

Lit des signaux depuis un fichier au taux-k.

## Syntaxe

```
fink ifilename, iskipframes, iformat, kin1 [, kin2] [, kin3] [...]
```

## Initialisation

*ifilename* -- nom du fichier d'entrée (peut être une chaîne de caractères ou un identificateur numérique généré par *fiopen*).

*iskipframes* -- nombre de trames à ignorer au début (chaque trame contient un échantillon de chaque canal).

*iformat* -- un nombre spécifiant le format du fichier d'entrée. Si un en-tête est trouvé, cet argument est ignoré.

- 0 - flottants sur 32 bit sans en-tête
- 1 - entiers sur 16 bit sans en-tête

## Exécution

*fink* est semblable à *fin* mais il opère au taux-k.



### Note

Prière de noter que comme cet opcode génère sa sortie en utilisant des paramètres d'entrée (placés à droite de l'opcode), ces variables doivent avoir été initialisées avant leur utilisation, sinon une erreur "utilisé avant d'être défini" se produira. On peut utiliser l'opcode *init* pour cela.

## Voir Aussi

*fin*, *fini*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound



# fiopen

fiopen — Ouvre un fichier dans un mode spécifique.

## Description

On peut utiliser *fiopen* pour ouvrir un fichier dans un des modes spécifiés.

## Syntaxe

```
ihandle fiopen ifilename, imode
```

## Initialisation

*ihandle* -- un nombre qui spécifie ce fichier.

*ifilename* -- le nom du fichier de sortie (entre guillemets).

*imode* -- choix du mode d'ouverture du fichier. *imode* peut prendre une des valeurs suivantes :

- 0 - ouvre un fichier texte en écriture
- 1 - ouvre un fichier texte en lecture
- 2 - ouvre un binaire texte en écriture
- 3 - ouvre un fichier binaire en lecture

## Exécution

*fiopen* ouvre un fichier à utiliser par la famille d'opcodes *fout*. Il est plus sûr de l'utiliser dans la section d'en-tête, en dehors de tout instrument. Il retourne un nombre, *ihandle*, qui fait référence de manière univoque au fichier ouvert.

Si *fiopen* est appelé sur un fichier déjà ouvert, il retourne simplement le même identificateur, et ne ferme pas le fichier.

Noter que *fout* et *foutk* peuvent utiliser soit un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.

## Voir Aussi

*ficlose fout, fouti, foutir, foutk*

## Crédits

Auteur : Gabriel Maldonado  
Italie

1999

Nouveau dans la version 3.56 de Csound

# flanger

flanger — A user controlled flanger.

## Description

A user controlled flanger.

## Syntax

```
ares flanger asig, adel, kfeedback [, imaxd]
```

## Initialization

*imaxd*(optional) -- maximum delay in seconds (needed for initial memory allocation)

## Performance

*asig* -- input signal

*adel* -- delay in seconds

*kfeedback* -- feedback amount (in normal tasks this should not exceed 1, even if bigger values are allowed)

This unit is useful for generating choruses and flangers. The delay must be varied at a-rate connecting *adel* to an oscillator output. Also the feedback can vary at k-rate. This opcode is implemented to allow *kr* different than *sr* (else delay could not be lower than *ksmps*) enhancing realtime performance. This unit is very similar to *wguide1*, the only difference is *flanger* does not have the lowpass filter.

## Examples

Here is an example of the flanger opcode. It uses the file *flanger.csd* [examples/flanger.csd], and *beats.wav* [examples/beats.wav].

### Exemple 156. Example of the flanger opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o flanger.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```

nchnls = 1

; Instrument #1.
instr 1
; Use the "beat.wav" audio file.
asig soundin "beats.wav"

; Vary the delay amount from 0 to 0.01 seconds.
adel line 0, p3, 0.01
kfeedback = 0.7

; Apply flange to the input signal.
aflang flanger asig, adel, kfeedback

; It can get loud, so clip its amplitude to 30,000.
al clip aflang, 1, 30000
out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Gabriel Maldonado  
Italy

Example written by Kevin Conder.

New in Csound version 3.49

# flashtxt

flashtxt — Permet d'afficher du text depuis des instruments sous la forme de curseurs.

## Description

Permet d'afficher du text depuis des instruments sous la forme de curseurs, etc. (Actuellement, ne fonctionne que sous Unix et Windows).

## Syntaxe

```
flashtxt iwhich, String
```

## Initialisation

*iwhich* -- le numéro de la fenêtre.

*String* -- la chaîne de caractères à afficher.

## Exécution

Une fenêtre est créée, identifiée par l'argument *iwhich*, avec la chaîne de caractères affichée. Si le texte est remplacé par un nombre, la fenêtre est effacée. Noter que les fenêtres de texte sont numérotées globalement si bien que différents instruments peuvent changer le texte, et que la fenêtre survit à l'instance de l'instrument.

## Exemples

Voici un exemple de l'opcode flashtxt. Il utilise le fichier *flashtxt.csd* [examples/flashtxt.csd].

### Exemple 157. Exemple de l'opcode flashtxt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o flashtxt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1
  flashtxt 1, "Instr 1 live"
  ao oscil 4000, 440, 1
  out ao
```

endin

```
</CsInstruments>  
<CsScore>
```

```
; Table 1: an ordinary sine wave.  
f 1 0 32768 10 1
```

```
; Play Instrument #1 for three seconds.  
i 1 0 3  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

# FLbox

FLbox — Un widget FLTK qui affiche du texte dans une boîte.

## Description

Un widget FLTK qui affiche du texte dans une boîte.

## Syntaxe

```
ihandle FLbox "label", itype, ifont, isize, iwidth, iheight, ix, iy [, image]
```

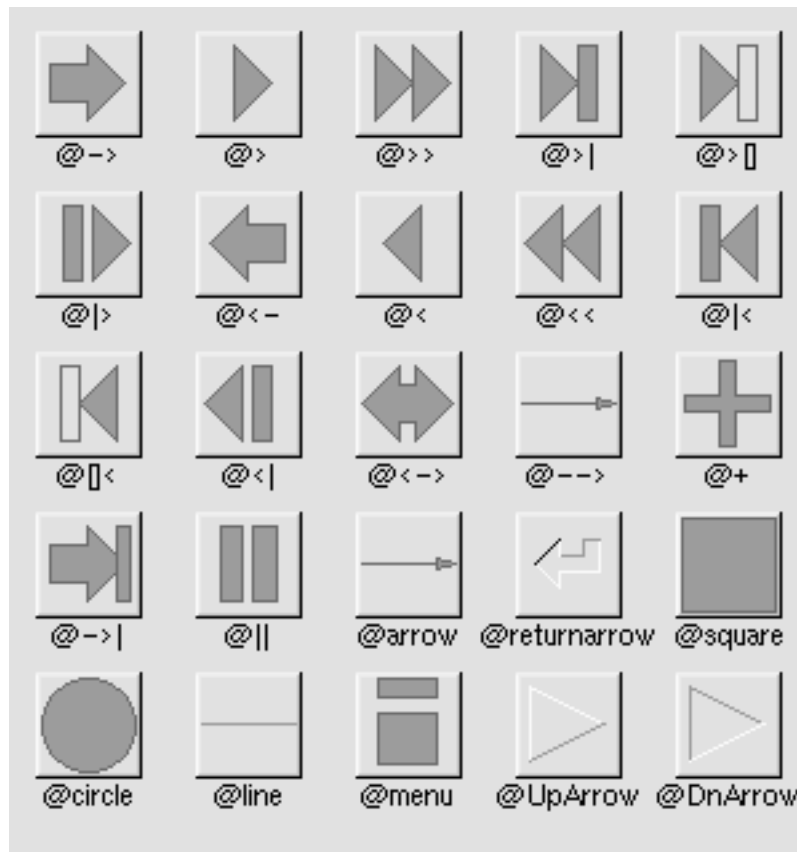
## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLbox* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

Noter qu'avec *FLbox* il n'est pas nécessaire d'appeler l'opcode *FLsetTextType* pour utiliser un symbole. Dans ce cas, il suffit d'utiliser une étiquette commençant par « @ » suivi de la chaîne de formatage correcte.

Les symboles suivants sont supportés :



Symboles d'étiquette FLTK supportés.

Le signe @ peut être suivi par les caractères de « formatage » facultatifs suivants, dans cet ordre :

1. « # » force une image carrée sans distortion de la forme du widget.
2. +[1-9] ou -[1-9] grossit ou diminue l'image.
3. [1-9] effectue une rotation d'un multiple de 45 degrés. « 6 » ne fait rien, les autres valeurs pointent dans la direction de cette touche sur un pavé numérique.

*itype* -- un nombre entier dénotant l'apparence du widget. Les valeurs suivantes sont acceptées :

- 1 - boîte sans relief
- 2 - boîte saillante
- 3 - boîte en creux
- 4 - boîte légèrement saillante
- 5 - boîte légèrement en creux
- 6 - boîte gravée
- 7 - boîte en relief



- 8 - boîte avec cadre
- 9 - boîte ombrée
- 10 - boîte arrondie
- 11 - boîte arrondie ombrée
- 12 - boîte arrondie sans relief
- 13 - boîte arrondie saillante
- 14 - boîte arrondie creuse
- 15 - boîte en losange saillante
- 16 - boîte en losange en creux
- 17 - boîte ovale
- 18 - boîte ovale ombrée
- 19 - boîte ovale sans relief

*ifont* -- un nombre entier dénotant le type de la police de *FLbox*. Les valeurs suivantes sont acceptées :

- 1 - helvetica (comme "Arial" sous Windows)
- 2 - helvetica gras
- 3 - helvetica italique
- 4 - helvetica gras italique
- 5 - courrier
- 6 - courrier gras
- 7 - courrier italique
- 8 - courrier gras italique
- 9 - times
- 10 - times gras
- 11 - times italique
- 12 - times gras italique
- 13 - symbol
- 14 - screen
- 15 - screen gras
- 16 - dingbats

*isize* -- taille de la police.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*image* -- un identifiant faisant référence à une image éventuellement ouverte avec l'opcode *bmopen*. S'il est utilisé, cela permet un skin pour ce widget.



### Note sur l'opcode *bmopen*

Bien qu'il soit mentionné, l'opcode *bmopen* n'a pas été implémenté dans Csound 4.22.

## Exécution

*FLbox* est utile pour montrer du texte dans une fenêtre. Le texte est à l'intérieur d'une boîte dont l'aspect dépend de l'argument *itype*.

Noter que *FLbox* n'est pas un valuateur et que sa valeur est constante. Elle ne peut pas être modifiée.

## Exemples

Voici un exemple de l'opcode *FLbox*. Il utilise le fichier *FLbox.csd* [examples/FLbox.csd].

### Exemple 158. Exemple de l'opcode *FLbox*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLbox.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Text Box", 700, 400, 50, 50
; Box border type (7=embossed box)
itype = 7
; Font type (10='Times Bold')
ifont = 10
; Font size
isize = 20
; Width of the flbox
iwidth = 400
; Height of the flbox
iheight = 30
```

```
    ; Distance of the left edge of the flbox
    ; from the left edge of the panel
    ix = 150
    ; Distance of the upper edge of the flbox
    ; from the upper edge of the panel
    iy = 100

    ih3 FLbox "Use Text Boxes For Labelling", itype, ifont, isize, iwidth, iheight, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
endin

</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLbutBank, FLbutton, FLprintk, FLprintk2, FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLbutBank

FLbutBank — Un opcode de widget FLTK qui crée un banc de boutons.

## Description

Un opcode de widget FLTK qui crée un banc de boutons.

## Syntaxe

```
kout, ihandle FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy, \  
iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [....] [, kpN]
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLbutBank* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

*itype* -- un nombre entier dénotant l'apparence du widget. Les nombres acceptés sont :

- 1 - bouton normal
- 2 - bouton lumineux
- 3 - bouton à cocher
- 4 - bouton avec un cercle à cocher

On peut ajouter 20 à la valeur pour créer un bouton de type "plastique". Noter qu'il n'y a pas de bouton arrondi plastique (si vous fixer le type à 24, le bouton aura la même apparence qu'avec le type 23).

*inumx* -- nombre de boutons dans chaque rangée du banc.

*inumy* -- nombre de boutons dans chaque colonne du banc.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante, exprimée en pixels.

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante, exprimée en pixels.

*iopcode* -- type de l'instruction de partition. Il faut fournir le code ASCII de la lettre correspondant à l'instruction de partition. Actuellement seules les instructions de partition « i » (code ASCII 105) sont supportées. Une valeur de zéro fait référence à une valeur de « i » par défaut. Ainsi 0 et 105 activent l'instruction *i*. Une valeur de -1 désactive cette possibilité.

## Exécution

*kout* -- valeur de sortie.

*kp1*, *kp2*, ..., *kpN* -- arguments des instruments activés.

L'opcode *FLbutBank* crée un banc de boutons. Par exemple, la ligne suivante :

```
gkButton,ihbl FLbutBank 22, 8, 8, 380, 180, 50, 350, 0, 7, 0, 0, 5000, 6000
```

créera ce banc :



FLbutBank.

Un clic sur un bouton coche celui-ci. Il peut aussi décocher un bouton préalablement coché appartenant au même banc. Ces boutons se comportent donc toujours comme des boutons radio. Noter que chaque bouton est étiqueté avec un nombre croissant. L'argument *kout* reçoit ce nombre lorsque le bouton correspondant est coché.

Non seulement *FLbutBank* retourne une valeur, mais il peut aussi activer (ou programmer) un instrument fourni par l'utilisateur chaque fois qu'un bouton est cliqué. Si l'argument *iopcode* est fixé à un nombre négatif, aucun instrument n'est activé ; ainsi cette possibilité est facultative. Pour activer un instrument, *iopcode* doit valoir 0 ou 105 (le code ASCII du caractère « i », faisant référence à l'instruction de partition *i*). Les p-champs de l'instrument activé sont *kp1* (numéro de l'instrument), *kp2* (date de l'action), *kp3* (durée), suivis des autres p-champs.

L'argument *itype* fixe le type des boutons de la même manière que pour l'opcode *FLbutton*. En ajoutant 10 à l'argument *itype* (c'est à dire en lui attribuant la valeur 11 pour le type 1, 12 pour le type 2, 13 pour le type 3 et 14 pour le type 4), il est possible d'ignorer la valeur courante de *FLbutBank* lorsque l'on sauve ou que l'on récupère des instantanés (voir *Opcodes Généraux relatifs aux Widgets FLTK*). On peut aussi ajouter 10 aux type de boutons "plastiques" (31 pour le type 1, 32 pour le type 2, etc).

*FLbutBank* est très utile pour retrouver des instantanés.

## Exemples

Voici un exemple de l'opcode *FLbutBank*. Il utilise le fichier *FLbutBank.csd* [examples/FLbutBank.csd].

### Exemple 159. Exemple de l'opcode FLbutBank.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLbutton.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
nchnls = 1

FLpanel "Button Bank", 520, 140, 100, 100
;itype = 2      ;Light Buttons
itype = 22      ;Plastic Light Buttons
inumx = 10
inumy = 4
iwidth = 500
iheight = 120
ix = 10
iy = 10
iopcode = 0
istarttim = 0
idur = 1

gkbutton, ihbb FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy, iopcode, 1, istarttim, idur

FLpanelEnd
FLrun

instr 1
  ibutton = i(gkbutton)
  prints "Button %i pushed!\n", ibutton
endin

</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLbox, FLbutton, FLprintk, FLprintk2, FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLbutton

FLbutton — Un opcode de widget FLTK qui crée un bouton.

## Description

Un opcode de widget FLTK qui crée un bouton.

## Syntaxe

```
kout, ihandle FLbutton "label", ion, ioff, itype, iwidth, iheight, ix, \
      iy, iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [....] [, kpN]
```

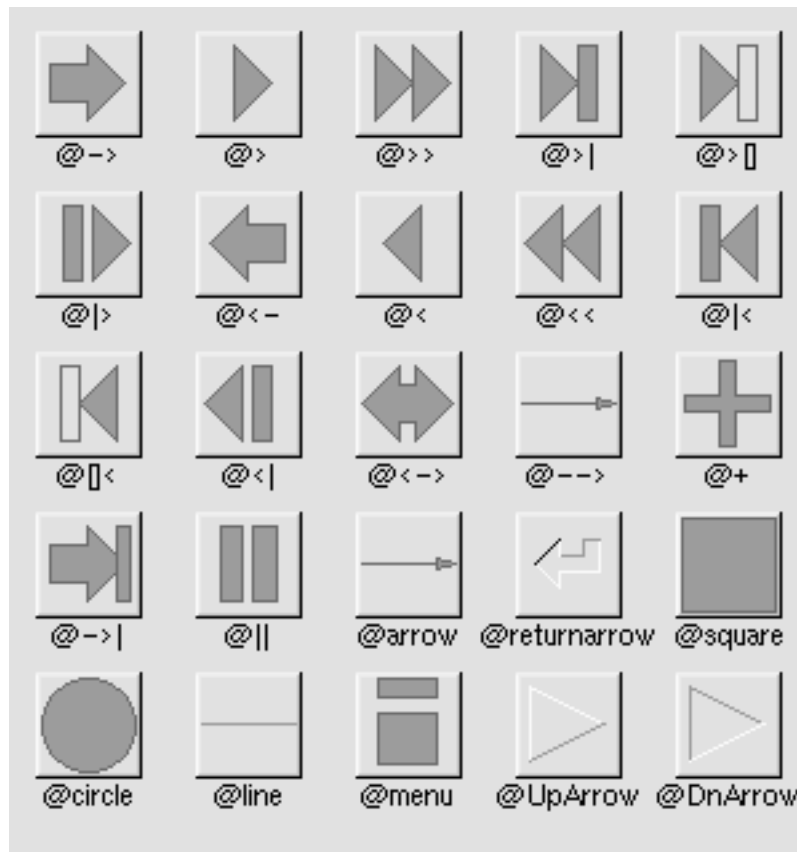
## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLbutton* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

Noter qu'avec *FLbutton* il n'est pas nécessaire d'appeler l'opcode *FLsetTextType* pour utiliser un symbole. Dans ce cas, il suffit d'utiliser une étiquette commençant par « @ » suivi de la chaîne de formatage correcte.

Les symboles suivants sont supportés :



Symboles d'étiquette FLTK supportés.

Le signe @ peut être suivi par les caractères de « formatage » facultatifs suivants, dans cet ordre :

1. « # » force une image carrée sans distortion de la forme du widget.
2. +[1-9] or -[1-9] grossit ou diminue l'image.
3. [1-9] effectue une rotation d'un multiple de 45 degrés. « 6 » ne fait rien, les autres valeurs pointent dans la direction de cette touche sur un pavé numérique.

*ion* -- valeur retournée quand le bouton est coché.

*ioff* -- valeur retournée quand le bouton est décoché.

*itype* -- un nombre entier dénotant l'apparence du widget.

Plusieurs types de bouton sont possibles selon la valeur de l'argument *itype* :

- 1 - bouton normal
- 2 - bouton lumineux
- 3 - bouton à cocher
- 4 - bouton avec un cercle à cocher



On peut ajouter 20 à la valeur pour créer un bouton de type "plastique". Noter qu'il n'y a pas de bouton arrondi plastique (si vous fixer le type à 24, le bouton aura la même apparence qu'avec le type 23).

Voici l'apparence des boutons :



FLbutton.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iopcode* -- type de l'instruction de partition. Il faut fournir le code ASCII de la lettre correspondant à l'instruction de partition. Actuellement seules les instructions de partition « i » (code ASCII 105) sont supportées. Une valeur de zéro fait référence à une valeur de « i » par défaut. Ainsi 0 et 105 activent l'instruction *i*. Une valeur de -1 désactive cette possibilité.

## Exécution

*kout* -- valeur de sortie.

*kp1, kp2, ..., kpN* -- arguments des instruments activés.

Les boutons de type 2, 3 et 4 retournent (argument *kout*) la valeur contenue dans l'argument *ion* lorsqu'ils sont cochés et celle contenue dans l'argument *ioff* lorsqu'ils sont décochés.

En ajoutant 10 à l'argument *itype* (c'est à dire en lui attribuant la valeur 11 pour le type 1, 12 pour le type 2, 13 pour le type 3 et 14 pour le type 4), il est possible d'ignorer la valeur du bouton lorsque l'on sauve ou que l'on récupère des instantanés (voir la section suivante). Non seulement *FLbutton* retourne une valeur, mais il peut aussi activer (ou programmer) un instrument fourni par l'utilisateur chaque fois le bou-

ton est cliqué. On peut aussi ajouter 10 aux type de boutons "plastiques" (31 pour le type 1, 32 pour le type 2, etc).

Si l'argument *iopcode* est fixé à un nombre négatif, aucun instrument n'est activé. Ainsi cette possibilité est facultative. Pour activer un instrument, *iopcode* doit valoir 0 ou 105 (le code ASCII du caractère « i », faisant référence à l'instruction de partition *i*).

Les p-champs de l'instrument activé sont *kp1* (numéro de l'instrument), *kp2* (date de l'action), *kp3* (durée), suivis des p-champs de l'utilisateur. Noter que pour les boutons à deux états (bouton lumineux, bouton à cocher, bouton avec un cercle à cocher), l'instrument n'est activé que lorsque le l'état du bouton passe de décoché à coché (pas de coché à décoché).

## Exemples

Voici un exemple de l'opcode FLbutton. Il utilise les fichiers *FLbutton.csd* [examples/FLbutton.csd] et *beats.wav* [examples/beats.wav].

### Exemple 160. Exemple de l'opcode FLbutton.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLbutton.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Using FLbuttons to create on screen controls for play,
; stop, fast forward and fast rewind of a sound file
; This example also makes use of a preset graphic for buttons.

sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

FLpanel "Buttons", 240, 400, 100, 100
  ion = 0
  ioff = 0
  itype = 1
  iwidth = 50
  iheight = 50
  ix = 10
  iy = 10
  iopcode = 0
  istarttim = 0
  idur = -1 ;Turn instruments on indefinitely

; Normal speed forwards
gkplay, ihb1 FLbutton "@>", ion, ioff, itype, iwidth, iheight, ix, iy, iopcode, 1, istarttim, idur,
; Stationary
gkstop, ihb2 FLbutton "@square", ion, ioff, itype, iwidth, iheight, ix+55, iy, iopcode, 2, istarttim, idur,
; Double speed backwards
gkrew, ihb3 FLbutton "@<<", ion, ioff, itype, iwidth, iheight, ix + 110, iy, iopcode, 1, istarttim, idur,
; Double speed forward
gkff, ihb4 FLbutton "@>>", ion, ioff, itype, iwidth, iheight, ix+165, iy, iopcode, 1, istarttim, idur,
; Type 1
gkt1, iht1 FLbutton "1-Normal Button", ion, ioff, 1, 200, 40, ix, iy + 65, -1
; Type 2
gkt2, iht2 FLbutton "2-Light Button", ion, ioff, 2, 200, 40, ix, iy + 110, -1
; Type 3
gkt3, iht3 FLbutton "3-Check Button", ion, ioff, 3, 200, 40, ix, iy + 155, -1
; Type 4
gkt4, iht4 FLbutton "4-Round Button", ion, ioff, 4, 200, 40, ix, iy + 200, -1
```

```

; Type 21
gkt5, iht5 FLbutton "21-Plastic Button", ion, ioff, 21, 200, 40, ix, iy + 245, -1
; Type 22
gkt6, iht6 FLbutton "22-Plastic Light Button", ion, ioff, 22, 200, 40, ix, iy + 290, -1
; Type 23
gkt7, iht7 FLbutton "23-Plastic Check Button", ion, ioff, 23, 200, 40, ix, iy + 335, -1
FLpanelEnd
FLrun

; Ensure that only 1 instance of instr 1
; plays even if the play button is clicked repeatedly
insnum = 1
icount = 1
maxalloc insnum, icount

instr 1
  asig disk2 "beats.wav", p4, 0, 1
  outs asig, asig
endin

instr 2
  turnoff2 1, 0, 0 ;Turn off instr 1
  turnoff ;Turn off this instrument
endin

</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLbox, FLbutBank, FLprintk, FLprintk2, FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLcloseButton

FLcloseButton — Un opcode de widget FLTK qui crée un bouton qui fermera la fenêtre du panneau auquel il appartient.

## Description

Un opcode de widget FLTK qui crée un bouton qui fermera la fenêtre du panneau auquel il appartient.

## Syntaxe

```
ihandle FLcloseButton "label", iwidth, iheight, ix, iy
```

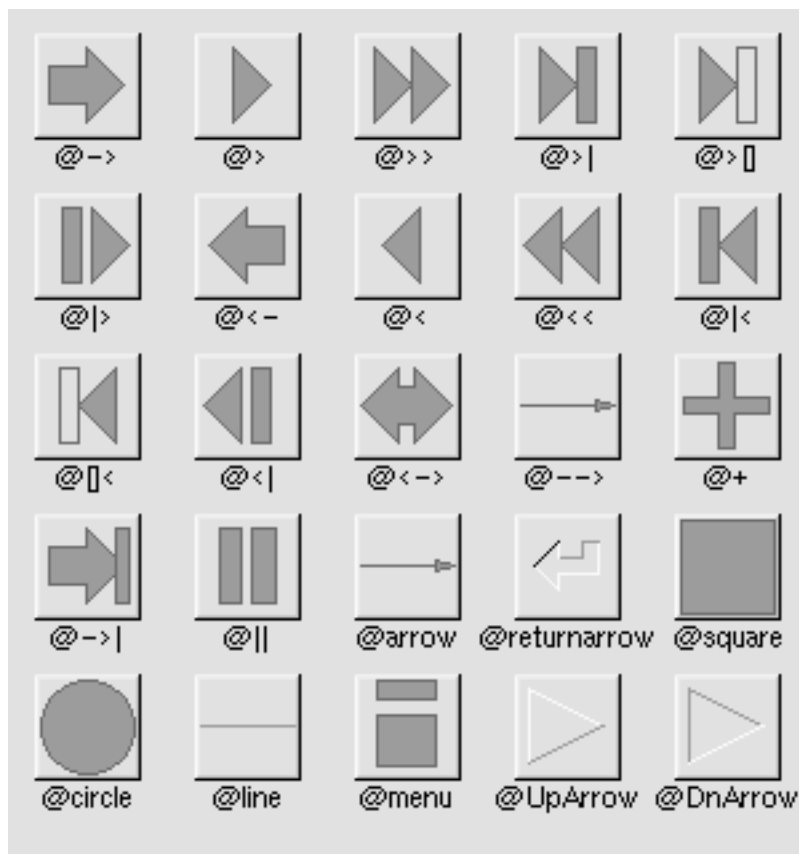
## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLcloseButton* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

Noter qu'avec *FLcloseButton* il n'est pas nécessaire d'appeler l'opcode *FLsetTextType* pour utiliser un symbole. Dans ce cas, il suffit d'utiliser une étiquette commençant par « @ » suivi de la chaîne de formatage correcte.

Les symboles suivants sont supportés :



Symboles d'étiquette FLTK supportés.

Le signe @ peut être suivi par les caractères de « formatage » facultatifs suivants, dans cet ordre :

1. « # » force une image carrée sans distortion de la forme du widget.
2. +[1-9] or -[1-9] grossit ou diminue l'image.
3. [1-9] effectue une rotation d'un multiple de 45 degrés. « 6 » ne fait rien, les autres valeurs pointent dans la direction de cette touche sur un pavé numérique.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Voir Aussi

*FLbutton*, *FLbox*, *FLbutBank*, *FLprintk*, *FLprintk2*, *FLvalue*

## Crédits

Auteur : Steven Yi

Nouveau dans la version 5.05

# FLcolor

FLcolor — Un opcode FLTK qui fixe les couleurs principales.

## Description

Fixe les couleurs principales à des valeurs RVB données par l'utilisateur.

## Syntaxe

```
FLcolor ired, igrreen, iblue [, ired2, igrreen2, iblue2]
```

## Initialisation

*ired* -- La composante rouge du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igrreen* -- La composante verte du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*ired2* -- La composante rouge de la couleur secondaire du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igrreen2* -- La composante verte de la couleur secondaire du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue2* -- La composante bleue de la couleur secondaire du widget cible. Chaque composante RVB est comprise entre 0 et 255.

## Exécution

Ces opcodes modifient l'apparence d'autres widgets. Ils sont de deux types : ceux ne contenant pas d'argument *ihandle* qui affectent tous les widgets déclarés après eux, et ceux ayant un argument *ihandle* qui n'affectent qu'un widget cible déclaré avant eux.

*FLcolor* fixe les couleurs principales à des valeurs RVB données par l'utilisateur. Cet opcode affecte la couleur principale de (presque) tous les widgets définis après lui. On peut placer plusieurs instances de *FLcolor* devant chaque widget que l'on veut modifier. Cependant, pour modifier un seul widget, il sera plus judicieux d'utiliser un opcode appartenant à la seconde catégorie (c'est-à-dire ceux contenant l'argument *ihandle*).

*FLcolor* est conçu pour modifier les couleurs d'un groupe de widgets en relation, supposés être de la même couleur. L'influence de *FLcolor* sur les widgets suivants peut-être désactivée en utilisant -1 comme seul argument de l'opcode. De plus, en utilisant -2 (ou -3) comme seul argument de *FLcolor*, tous les widgets suivants auront une couleur choisie aléatoirement. -2 sélectionne une couleur aléatoire claire, tandis que -3 sélectionne une couleur aléatoire foncée.

L'utilisation de *ired2*, *igrreen2*, *iblue2* est équivalente à l'utilisation d'un *FLcolor2* séparé.

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*,

*FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLcolor2

FLcolor2 — Un opcode FLTK qui fixe la couleur secondaire (de sélection).

## Description

*FLcolor2* est semblable à *FLcolor* sauf qu'il affecte la couleur secondaire (de sélection).

## Syntaxe

```
FLcolor2 ired, igreen, iblue
```

## Initialization

*ired* -- La composante rouge du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue du widget cible. Chaque composante RVB est comprise entre 0 et 255.

## Exécution

Ces opcodes modifient l'apparence d'autres widgets. Ils sont de deux types : ceux ne contenant pas d'argument *ihandle* qui affectent tous les widgets déclarés après eux, et ceux ayant un argument *ihandle* qui n'affectent qu'un widget cible déclaré avant eux.

*FLcolor2* est semblable à *FLcolor* sauf qu'il affecte la couleur secondaire (de sélection). L'influence de *FLcolor2* sur les widgets suivants peut-être désactivée en le fixant à -1. Avec une valeur de -2 (ou -3), tous les widgets suivants auront une couleur secondaire choisie aléatoirement. -2 sélectionne une couleur aléatoire claire, tandis que -3 sélectionne une couleur aléatoire foncée.

## Voir Aussi

*FLcolor*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLcount

FLcount — Un opcode de widget FLTK qui crée un compteur.

## Description

Permet à l'utilisateur d'augmenter/diminuer une valeur avec des clics de souris sur un bouton fléché correspondant.

## Syntaxe

```
kout, ihandle FLcount "label", imin, imax, istep1, istep2, itype, \  
    iwidth, iheight, ix, iy, iopcode [, kp1] [, kp2] [, kp3] [...] [, kpN]
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du valuateur. Il est automatiquement fixé par le valuateur.

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- valeur minimale de l'intervalle de sortie.

*imax* -- valeur maximale de l'intervalle de sortie.

*istep1* -- un nombre en virgule flottante indiquant le pas d'incréméntation du valuateur à chaque clic de souris. *istep1* sert aux réglages fins.

*istep2* -- un nombre en virgule flottante indiquant le pas d'incréméntation du valuateur à chaque clic de souris. *istep2* sert aux réglages grossiers.

*itype* -- un nombre entier dénotant l'apparence du valuateur.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

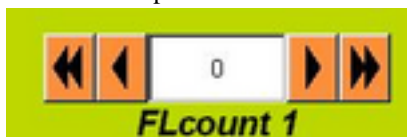
*iopcode* -- type de l'instruction de partition. Il faut fournir le code ASCII de la lettre correspondant à l'instruction de partition. Actuellement seules les instructions de partition « i » (code ASCII 105) sont supportées. Une valeur de zéro fait référence à une valeur de « i » par défaut. Ainsi 0 et 105 activent l'instruction *i*. Une valeur de -1 désactive cette possibilité.

## Exécution

*kout* -- valeur de sortie.

*kp1, kp2, ..., kpN* -- arguments des instruments activés.

*FLcount* permet à l'utilisateur d'augmenter/diminuer une valeur avec des clics de souris sur les boutons fléchés correspondants :



FLcount.

Il y a deux sortes de boutons fléchés, pour des pas plus grands ou plus petits. Noter que non seulement *FLcount* retourne une valeur et un identifiant, mais il peut aussi activer (programmer) un instrument fourni par l'utilisateur chaque fois qu'un bouton est cliqué. Les p-champs de l'instrument activé sont *kp1* (numéro de l'instrument), *kp2* (date de l'action), *kp3* (durée) suivis des p-champs de l'utilisateur. Si l'argument *iopcode* est fixé à un nombre négatif, aucun instrument n'est activé. Ainsi cette possibilité est facultative.

## Exemples

Voici un exemple de l'opcode *FLcount*. Il utilise le fichier *FLcount.csd* [exemples/FLcount.csd].

### Exemple 161. Exemple de l'opcode *FLcount*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLcount.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flcount opcode
; clicking on the single arrow buttons
; increments the oscillator in semitone steps
; clicking on the double arrow buttons
; increments the oscillator in octave steps
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Counter", 900, 400, 50, 50
; Minimum value output by counter
imin = 6
; Maximum value output by counter
imax = 12
; Single arrow step size (semitones)
istep1 = 1/12
; Double arrow step size (octave)
istep2 = 1
; Counter type (1=double arrow counter)
itype = 1
; Width of the counter in pixels
iwidth = 200
; Height of the counter in pixels
iheight = 30
; Distance of the left edge of the counter
; from the left edge of the panel
ix = 50
; Distance of the top edge of the counter
; from the top edge of the panel
iy = 50
```

```

; Score event type (-1=ignored)
iopcode = -1

gkoct, ihandle FLcount "pitch in oct format", imin, imax, istep1, istep2, itype, iwidth, iheight, i
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, cpsoct(gkoct), ifn
    out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLjoy, FLknob, FLroller, FLslider, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLexecButton

FLexecButton — Un opcode de widget FLTK qui crée un bouton qui exécute une commande.

## Description

Un opcode de widget FLTK qui crée un bouton qui exécute une commande. Utile pour ouvrir une documentation HTML comme un texte Au sujet de ou pour démarrer un programme séparé depuis une interface de widgets FLTK.



### Avertissement

Comme n'importe quelle commande peut être exécutée, il faut être très prudent en utilisant cet opcode et en exécutant des orchestres d'autres personnes utilisant cet opcode.

## Syntaxe

```
ihandle FLexecButton "command", iwidth, iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLexecButton*.

« *command* » -- une chaîne de caractères entre guillemets contenant la commande à exécuter.

Noter qu'avec *FLexecButton*, le texte par défaut du bouton est "About" et qu'il est nécessaire d'appeler l'opcode *FLsetText* pour changer le texte du bouton.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exemples

Voici un exemple de l'opcode *FLexecButton*. Il utilise le fichier *FLexecButton.csd* [exemples/FLexecButton.csd].

### Exemple 162. Exemple de l'opcode FLexecButton.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No display
-odac         -iadc      -d          ;;RT audio I/O
</CsOptions>
<CsInstruments>

    sr      = 44100
    ksmps   = 10
    nchnls  = 1

; Example by Jonathan Murphy 2007

;;; reset amplitude range
0dbfs      = 1

;;; set the base colour for the panel
FLcolor    100, 0, 200
;;; define the panel
FLpanel    "FExecButton", 250, 100, 0, 0
;;; sliders to control time stretch and pitch
gkstr, gistretch FLslider    "Time", 0.5, 1.5, 0, 6, -1, 10, 60, 150, 20
gkpch, gipitch  FLslider    "Pitch", 0.5, 1.5, 0, 6, -1, 10, 60, 200, 20
;;; set FExecButton colour
FLcolor    255, 255, 0
;;; when this button is pressed, fourier analysis is performed on the file
;;; "beats.wav", producing the analysis file "beats.pvx"
gipvoc     FExecButton    "csound -U pvanal beats.wav beats.pvx", 60, 20, 20, 20
;;; set FExecButton text
FLsetText  "PVOC", gipvoc
;;; when this button is pressed, instr 10000 is called, exiting
;;; Csound immediately

;;; cancel previous colour
FLcolor    -1
;;; set colour for kill button
FLcolor    255, 0, 0
gkkill, gikill FLbutton    "X", 1, 1, 1, 20, 20, 100, 20, 0, 10000, 0, 0.1
;;; cancel previous colour
FLcolor    -1
;;; set colour for play/stop and pause buttons
FLcolor    0, 200, 0
;;; pause and play/stop buttons
gkpause, gipause FLbutton    "@|", 1, 0, 2, 40, 20, 20, 60, -1
gkplay, gipplay  FLbutton    "@|>", 1, 0, 2, 40, 20, 80, 60, -1
;;; end the panel
FLpanelEnd
;;; set initial values for time stretch and pitch
FLsetVal_i 1, gistretch
FLsetVal_i 1, gipitch
;;; run the panel
FLrun

    instr 1                                ; trigger play/stop
    ;; is the play/stop button on or off?
    ;; either way we need to trigger something,
    ;; so we can't just use the value of gkplay
    kon     trigger    gkplay, 0, 0
    koff    trigger    gkplay, 1, 1
    ;; if on, start instr 2
    schedkwhen kon, -1, -1, 2, 0, -1
    ;; if off, stop instr 2
    schedkwhen koff, -1, -1, -2, 0, -1

    endin

    instr 2

    ;; paused or playing?
    if (gkpause == 1) kgoto pause
    kgoto start
pause:
    ;; if the pause button is on, skip sound production
    kgoto end
start:
    ;; get the length of the analysis file in seconds
    ilen    filelen    "beats.pvx"

```

```

    ;;; determine base frequency of playback
    icps      = 1/ilen
    ;;; create a table over the length of the file
    itpt      ftgen      0, 0, 513, -7, 0, 512, ilen
    ;;; phasor for time control
    kphs      phasor     icps * gkstr
    ;;; use phasor as index into table
    kndx      = kphs * 512
    ;;; read table
    ktpt      tablei     kndx, itpt
    ;;; use value from table as time pointer into file
    fsig1      pvsfread   ktpt, "beats.pvx"
    ;;; change playback pitch
    fsig2      pvscale    fsig1, gkpch
    ;;; resynthesize
    aout       pvsynth    fsig2
    ;;; envelope to avoid clicks and clipping
    aenv       linsegr    0, 0.3, 0.75, 0.1, 0
    aout       = aout * aenv
              out         aout
end:

    endin

    instr 10000                                ; kill

    exitnow

    endin

</CsInstruments>
<CsScore>
i1 0 10000
e
</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLbutton, FLbox, FLbutBank, FLprintk, FLprintk2, FLvalue*

## Crédits

Auteur : Steven Yi

Exemple par Jonathan Murphy

Nouveau dans la version 5.05

# FLgetsnap

FLgetsnap — Retrouve un instantané FLTK antérieurement enregistré.

## Description

Retrouve un instantané FLTK antérieurement enregistré (en mémoire), c'est-à-dire fixe tous les valeurs aux valeurs correspondantes enregistrées dans l'instantané.

## Syntaxe

```
inumsnap FLgetsnap index [, igroup]
```

## Initialisation

*inumsnap* -- nombre courant d'instantanés.

*index* -- un nombre faisant référence de manière univoque à un instantané. Plusieurs instantanés peuvent être enregistrés dans la même banque.

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valeurs. La valeur par défaut est zéro qui fait référence au premier groupe. Le numéro de groupe est déterminé par l'opcode *FLsetSnapGroup*.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

*FLgetsnap* retrouve un instantané FLTK antérieurement enregistré (en mémoire), c'est-à-dire fixe tous les valeurs aux valeurs correspondantes enregistrées dans l'instantané. L'argument *index* doit faire référence de manière univoque à un instantané existant. Si l'argument *index* fait référence à un instantané vide ou à un instantané qui n'existe pas, rien ne se produit. *FLsetsnap* retourne le nombre courant d'instantanés (argument *inumsnap*).

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

## Voir Aussi

*FLloadsnap*, *FLrun*, *FLsavesnap*, *FLsetsnap*, *FLsetSnapGroup*, *FLupdate*

## Crédits

Auteur : Gabriel Maldonado



Nouveau dans la version 4.22

# FLgroup

FLgroup — Un opcode de conteneur FLTK qui regroupe des widgets enfants.

## Description

Un opcode de conteneur FLTK qui regroupe des widgets enfants.

## Syntaxe

```
FLgroup "label", iwidth, iheight, ix, iy [, iborder] [, image]
```

## Initialisation

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iborder* (facultatif, 0 par défaut) -- type de la bordure du conteneur. Il est exprimé par un nombre entier choisi parmi les suivants :

- 0 - pas de bordure
- 1 - bordure de boîte en creux
- 2 - bordure de boîte saillante
- 3 - bordure gravée
- 4 - bordure en relief
- 5 - bordure ligne noire
- 6 - mince bordure en creux
- 7 - mince bordure saillante

Si le nombre entier ne correspond à aucune de ces valeurs, aucune bordure n'est fournie par défaut.

*image* (facultatif) -- un identifiant faisant référence à une image éventuellement ouverte avec l'opcode *bmopen*. S'il est utilisé, cela permet un skin pour ce widget.



### Note sur l'opcode *bmopen*

Bien qu'il soit mentionné, l'opcode *bmopen* n'a pas été implémenté dans Csound 4.22.

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphique des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de type-k dans les conteneurs.

## Voir Aussi

*FLgroupEnd, FLpack, FLpackEnd, FLpanel, FLpanelEnd, FLscroll, FLscrollEnd, FLtabs, FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLgroupEnd

FLgroupEnd — Marque la fin d'un groupe de widgets FLTK enfants.

## Description

Marque la fin d'un groupe de widgets FLTK enfants.

## Syntaxe

`FLgroupEnd`

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir Aussi

*FLgroup*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLgroupEnd

FLgroup\_end — Marque la fin d'un groupe de widgets FLTK enfants.

## Description

Marque la fin d'un groupe de widgets FLTK enfants. C'est un autre nom pour **FLgroupEnd** fourni pour des raisons de compatibilité. Voir *FLgroupEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLhide

FLhide — Cache le widget FLTK cible.

## Description

Cache le widget FLTK cible, le rendant invisible.

## Syntaxe

```
FLhide ihandle
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*).

## Exécution

*FLhide* cache le widget FLTK cible, le rendant invisible.

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLhvsBox

**FLhvsBox** — Affiche une boîte avec une grille utile pour visualiser la Synthèse Hyper Vectorielle à deux dimensions.

## Description

*FLhvsBox* affiche une boîte avec une grille utile pour visualiser la Synthèse Hyper Vectorielle à deux dimensions.

## Syntaxe

```
ihandle FLhvsBox inumlinesX, inumlinesY, iwidth, iheight, ix, iy [, image]
```

## Initialisation

*ihandle* – un identifiant (un nombre entier) défini univoquement pour identifier une boîte HVS spécifique (voir ci-dessous).

*inumlinesX, inumlinesY* - nombre de lignes verticales et horizontales délimitant les zones carrées HVS.

*iwidth, iheight* - largeur et hauteur de la boîte HVS.

*ix, iy* - position de la boîte HVS.

*image* – (facultatif, 0 par défaut) un nombre entier dénotant un image RVB ouverte avec l'opcode *bmo-pen*. Un zéro indique pas d'image.

## Exécution

*FLhvsBox* est un widget capable de montrer la position courante du curseur HVS dans une boîte HVS (c'est-à-dire une zone carrée contenant un grille). Le nombre de lignes horizontales et verticales de la grille peut être défini avec les arguments *inumlinesX, inumlinesY*. Cet opcode doit être déclaré dans un bloc *FLpanel* - *FLpanelEnd*. Voir l'entrée de l'opcode *hvs2* pour un exemple d'utilisation de *FLhvsBox*.

*FLhvsBoxSetValue* est utilisé pour fixer la position du curseur d'un widget *FLhvsBox*.



### Note

L'opcode *bmscan* n'a pas encore été implémenté, si bien que le paramètre *image* n'a actuellement pas d'effet.

## Voir Aussi

*hvs2, FLhvsBoxSetValue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLhvsBoxSetValue

FLhvsBoxSetValue — Fixe la position du curseur d'un widget *FLhvsBox* préalablement déclaré.

## Description

*FLhvsBoxSetValue* fixe la position du curseur d'un widget *FLhvsBox* préalablement déclaré.

## Syntaxe

**FLhvsBox** *kx*, *ky*, *ihandle*

## Initialisation

*ihandle* – un identifiant (un nombre entier) défini univoquement pour identifier une boîte HVS spécifique (voir ci-dessous).

## Exécution

*kx*, *ky* – les coordonnées de la position du curseur HVS à fixer.

*FLhvsBoxSetValue* fixe la position du curseur d'un widget *FLhvsBox* préalablement déclaré. Les arguments *kx* et *ky* dénotant la position du curseur doivent être exprimées en valeurs normalisées (comprises entre 0 et 1).

Voir l'entrée *hvs2* pour un exemple d'utilisation de *FLhvsBoxSetValue*.

## Voir Aussi

*hvs2*, *FLhvsBox*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06



# FLjoy

FLjoy — Un opcode FLTK qui agit comme un joystick.

## Description

*FLjoy* est une zone carrée qui permet à l'utilisateur de modifier deux valeurs de sortie en même temps. Il agit comme un joystick.

## Syntaxe

```
koutx, kouty, ihandlex, ihandley FLjoy "label", iminx, imaxx, iminy, \
    imaxy, iexpx, iexpy, idispx, idispy, iwidth, iheight, ix, iy
```

## Initialisation

*ihandlex* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du valuateur. Il est automatiquement fixé par le valuateur.

*ihandley* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du valuateur. Il est automatiquement fixé par le valuateur.

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*iminx* -- valeur x minimale de l'intervalle de sortie.

*imaxx* -- valeur x maximale de l'intervalle de sortie.

*iminy* -- valeur y minimale de l'intervalle de sortie.

*imaxy* -- valeur y maximale de l'intervalle de sortie.

*idispx* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*idispy* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*iexpx* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexpx* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.

*iexpy* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexpy* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.



## IMPORTANT !

Noter que les tables utilisées par les valuateurs doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En fait, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*koutx* -- valeur x en sortie.

*kouty* -- valeur y en sortie.

## Exemples

Voici un exemple de l'opcode *FLjoy*. Il utilise le fichier *FLjoy.csd* [examples/FLjoy.csd].

### Exemple 163. Exemple de l'opcode *FLjoy*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLjoy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flpanel opcode
; Horizontal click-dragging controls the frequency of the oscillator
; Vertical click-dragging controls the amplitude of the oscillator
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "X Y Panel", 900, 400, 50, 50
```

```

; Minimum value output by x movement (frequency)
iminx = 200
; Maximum value output by x movement (frequency)
imaxx = 5000
; Minimum value output by y movement (amplitude)
iminy = 0
; Maximum value output by y movement (amplitude)
imaxy = 15000
; Logarithmic change in x direction
iexpx = -1
; Linear change in y direction
iexpy = 0
; Display handle x direction (-1=not used)
idispx = -1
; Display handle y direction (-1=not used)
idispy = -1
; Width of the x y panel in pixels
iwidth = 800
; Height of the x y panel in pixels
iheight = 300
; Distance of the left edge of the x y panel from
; the left edge of the panel
ix = 50
; Distance of the top edge of the x y
; panel from the top edge of the panel
iy = 50

gkfreqx, gkampy, ihandlex, ihandley FLjoy "X - Frequency Y - Amplitude", iminx, imaxx, iminy, imaxy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    ifn = 1
    asig oscili gkampy, gkfreqx, ifn
    out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLcount, FLknob, FLroller, FLslider, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLkeyIn

**FLkeyIn** — Retourne les touches enfoncées (sur le clavier alphanumérique) quand un panneau FLTK est actif.

## Description

*FLkeyIn* nous informe sur l'état d'une touche enfoncée par l'utilisateur sur le clavier alphanumérique quand un panneau FLTK est actif.

## Syntaxe

```
kascii FLkeyIn [ifn]
```

## Initialisation

*ifn* – (facultatif, zéro par défaut) fixe le comportement de *FLkeyIn* (voir ci-dessous).

## Exécution

*kascii* - la valeur ASCII de la dernière touche enfoncée. Si la touche est enfoncée, la valeur est positive alors que si la touche est relâchée la valeur est négative.

*FLkeyIn* est utile pour savoir si une touche a été enfoncée sur le clavier de l'ordinateur. Le comportement de cet opcode dépend de l'argument facultatif *ifn*.

Si *ifn* = 0 (par défaut), *FLkeyIn* retourne le code ASCII de la dernière touche enfoncée. Si c'est une touche spéciale (ctrl, maj, alt, f1-f12, etc.), une valeur de 256 est ajoutée à la valeur retournée afin de la distinguer des touches normales. La sortie continuera à retourner la valeur de la dernière touche jusqu'à ce qu'une nouvelle touche soit enfoncée ou relâchée. Noter que la sortie sera négative lorsqu'une touche est relâchée.

Si *ifn* contient le numéro d'une table déjà allouée ayant au moins 512 éléments, l'élément de la table d'indice égal au code ASCII de la touche enfoncée est fixé à 1, tous les autres éléments de la table étant fixés à 0. Cela permet de tester l'état d'une touche particulière ou d'un ensemble de touches.

Noter qu'il faut que le paramètre *ikbdcapture* du *FLpanel* concerné doit être différent de 0 pour que *FLkeyIn* capture les événements de clavier provenant de ce panneau.



### Note

Comme *FLkeyIn* travaille en interne au taux-k, on ne peut pas l'utiliser dans l'en-tête comme les autres opcodes FLTK. On doit l'utiliser dans un instrument.

## Exemples

Voici un exemple de l'opcode *FLkeyIn*. Il utilise le fichier *FLkeyIn.csd* [exemples/FLkeyIn.csd].

### Exemple 164. Exemple de l'opcode *FLkeyIn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera 2007

FLpanel "FLkeyIn", 400, 300, -1, -1, 5, 1, 1
FLpanelEnd

FLrun

0dbfs = 1

instr 1
kascii  FLkeyIn
ktrig  changed kascii
if (kascii > 0) then
  printf "Key Down: %i\n", ktrig, kascii
else
  printf "Key Up: %i\n", ktrig, -kascii
endif
endin

</CsInstruments>
<CsScore>
i 1 0 120
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLknob

FLknob — Un opcode de widget FLTK qui crée un bouton rotatif.

## Description

Un opcode de widget FLTK qui crée un bouton rotatif.

## Syntaxe

```
kout, ihandle FLknob "label", imin, imax, iexp, itype, idisp, iwidth, \
ix, iy [, icursorsize]
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLknob* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- valeur minimale de l'intervalle de sortie.

*imax* -- valeur maximale de l'intervalle de sortie.

*iexp* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexp* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.



### IMPORTANT !

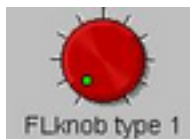
Noter que les tables utilisées par les valuateurs doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En fait, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

*itype* -- un nombre entier indiquant l'apparence du valuateur.

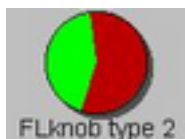
L'argument *itype* accepte les valeurs suivantes :

- 1 - un bouton rotatif 3D
- 2 - un bouton à secteur circulaire

- 3 - un bouton comme une horloge
- 4 - un bouton rotatif plat



Un bouton rotatif 3D.



Un bouton à secteur circulaire.



Un bouton comme une horloge.



Un bouton rotatif plat.

*idisp* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*icursorsize* (facultatif) -- Si *itype* de *FLknob* vaut 1 (bouton 3D), ce paramètre contrôle la taille du curseur du bouton.

## Exécution

*kout* -- valeur en sortie.

*FLknob* met un bouton rotatif dans le conteneur correspondant.

## Exemples

Voici un exemple de l'opcode FLknob. Il utilise le fichier *FLknob.csd* [examples/FLknob.csd].

### Exemple 165. Exemple de l'opcode FLknob.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d           ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLknob.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flknob controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Knob", 900, 400, 50, 50
; Minimum value output by the knob
imin = 200
; Maximum value output by the knob
imax = 5000
; Logarithmic type knob selected
iexp = -1
; Knob graphic type (1=3D knob)
itype = 1
; Display handle (-1=not used)
idisp = -1
; Width of the knob in pixels
iwidth = 70
; Distance of the left edge of the knob
; from the left edge of the panel
ix = 70
; Distance of the top edge of the knob
; from the top of the panel
iy = 125

gkfreq, ihandle FLknob "Frequency", imin, imax, iexp, itype, idisp, iwidth, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

; Set the widget's initial value
FLsetVal_i 300, ihandle

instr 1
iamp = 15000
ifn = 1
asig oscili iamp, gkfreq, ifn
out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsSoundSynthesizer>
```



Voici un autre exemple de l'opcode FLknob, montrant les différents styles de boutons ainsi que l'usage de FLvalue pour afficher la valeur d'un bouton. Il utilise le fichier FLknob-2.csd [examples/FLknob-2.csd].

### Exemple 166. Exemple plus complexe de l'opcode FLknob.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLknob.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

;By Andres Cabrera 2007
FLpanel "Knob Types", 330, 230, 50, 50
; Distance of the left edge of the knob
; from the left edge of the panel
ix = 20
; Distance of the top edge of the knob
; from the top of the panel
iy = 20

;Create boxes that display a widget's value
ihandleA FLvalue "A", 60, 20, ix + 130, iy + 110
ihandleB FLvalue "B", 60, 20, ix + 220, iy + 110
ihandleC FLvalue "C", 60, 20, ix + 130, iy + 160
ihandleD FLvalue "D", 60, 20, ix + 220, iy + 160

; The four types of FLknobs
gkdummy1, ihandle1 FLknob "Type 1", 200, 5000, -1, 1, ihandleA, 70, ix, iy, 90
gkdummy2, ihandle2 FLknob "Type 2", 200, 5000, -1, 2, ihandleB, 70, ix + 100, iy
gkdummy3, ihandle3 FLknob "Type 3", 200, 5000, -1, 3, ihandleC, 70, ix + 200, iy
gkdummy4, ihandle4 FLknob "Type 4", 200, 5000, -1, 4, ihandleD, 70, ix, iy + 100
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

; Set the color of widgets
FLsetColor 20, 23, 100, ihandle1
FLsetColor 0, 123, 100, ihandle2
FLsetColor 180, 23, 12, ihandle3
FLsetColor 10, 230, 0, ihandle4

FLsetColor2 200, 230, 0, ihandle1
FLsetColor2 200, 0, 123, ihandle2
FLsetColor2 180, 180, 100, ihandle3
FLsetColor2 180, 23, 12, ihandle4

; Set the initial value of the widget
FLsetVal_i 300, ihandle1
FLsetVal_i 1000, ihandle2

instr 1
; Nothing here for now
endin

</CsInstruments>
<CsScore>

f 0 3600 ;Dummy table to make csound wait for realtime events
```

e

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*FLcount, FLjoy, FLroller, FLslider, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLlabel

FLlabel — Un opcode FLTK qui modifie l'apparence d'une étiquette de texte.

## Description

Modifie un ensemble de paramètres en relation avec l'apparence de l'étiquette de texte d'un widget (c'est-à-dire la taille, la police, l'alignement et la couleur du texte correspondant).

## Syntaxe

**FLlabel** *isize, ifont, ialign, ired, igreen, iblue*

## Initialisation

*isize* -- taille de la police du widget cible. Les valeurs normales sont de l'ordre de 15. Des nombres plus grands augmentent la taille de la police, tandis que des nombres plus petits la réduisent.

*ifont* -- fixe le type de police de l'étiquette d'un widget.

Les valeurs admises pour l'argument *ifont* sont :

- 1 - Helvética (comme Arial sous Windows)
- 2 - Helvética Gras
- 3 - Helvética Italique
- 4 - Helvética Gras Italique
- 5 - Courier
- 6 - Courier Gras
- 7 - Courier Italique
- 8 - Courier Gras Italique
- 9 - Times
- 10 - Times Grasd
- 11 - Times Italique
- 12 - Times Gras Italique
- 13 - Symbole
- 14 - Ecran
- 15 - Ecran Gras
- 16 - Dingbats

*ialign* -- fixe l'alignement de l'étiquette de texte du widget.

Les valeurs admises pour l'argument *ialign* sont :

- 1 - centré
- 2 - en haut
- 3 - en bas
- 4 - à gauche
- 5 - à droite
- 6 - en haut à gauche
- 7 - en haut à droite
- 8 - en bas à gauche
- 9 - en bas à droite

*ired* -- la couleur rouge du widget cible. L'intervalle pour chaque composante RVB va de 0 à 255.

*igreen* -- la couleur verte du widget cible. L'intervalle pour chaque composante RVB va de 0 à 255.

*iblue* -- la couleur bleue du widget cible. L'intervalle pour chaque composante RVB va de 0 à 255.

## Exécution

*FLlabel* modifie un ensemble de paramètres en relation avec l'apparence de l'étiquette de texte d'un widget, c'est-à-dire la taille, la police, l'alignement et la couleur du texte correspondant. Cet opcode affecte (presque) tous les widgets définis après lui. On peut placer plusieurs instances de *FLlabel* avant chaque widget que l'on veut modifier. Cependant, pour modifier un widget particulier, il vaut mieux utiliser l'opcode appartenant au second type (c'est-à-dire ceux ayant un argument *ihandle*).

l'influence de *FLlabel* sur le widget suivant peut être suspendue en lui donnant -1 comme seul argument. *FLlabel* est conçu pour modifier les attributs de texte d'un groupe de widgets relatifs.

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLloadsnap

FLloadsnap — Charge tous les instantanés dans la banque de mémoire de l'orchestre courant.

## Description

*FLloadsnap* charge tous les instantanés contenus dans un fichier dans la banque de mémoire de l'orchestre courant.

## Syntaxe

```
FLloadsnap "filename" [, igroup]
```

## Initialisation

*"filename"* -- une chaîne de caractères entre guillemets correspondant au fichier à partir duquel charger une banque d'instantanés.

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valueurs. La valeur par défaut est zéro qui fait référence au premier groupe. Le numéro de groupe est déterminé par l'opcode *FLsetSnapGroup*.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

*FLloadsnap* charge tous les instantanés contenus dans *filename* dans la banque de mémoire de l'orchestre courant.

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

## Voir Aussi

*FLgetsnap*, *FLrun*, *FLsetSnapGroup*, *FLsavesnap*, *FLsetsnap*, *FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLmouse

FLmouse — Retourne la position de la souris et l'état de ses trois boutons.

## Description

*FLmouse* retourne les coordonnées de la position de la souris dans un panneau FLTK et l'état de ses trois boutons.

## Syntaxe

*kx*, *ky*, *kb1*, *kb2*, *kb3* **FLmouse** [*imode*]

## Initialisation

*imode* – (facultatif, 0 par défaut) Détermine la façon de rapporter la position de la souris.

- 0 - Position absolue normalisée dans l'intervalle 0-1
- 1 - Position absolue en pixel brut
- 2 - Position en pixel brut, relative au panneau FLTK

## Exécution

*kx*, *ky* – les coordonnées de la souris, dont l'intervalle dépend de l'argument *imode* (voir ci-dessus).

*kb1*, *kb2*, *kb3* – les états des boutons de la souris, 1 lorsque le bouton correspondant est enfoncé, 0 lorsqu'il est relâché.

*FLmouse* retourne les coordonnées de la position de la souris dans un panneau FLTK et l'état de ses trois boutons. Les coordonnées peuvent être récupérées selon trois modes dépendant de la valeur de l'argument *imode* (voir ci-dessus). Les modes 0 et 1 retournent la position de la souris par rapport à l'écran complet (mode absolu), tandis que le mode 2 retourne la position en pixels dans un panneau FLTK. Noter que *FLmouse* n'est actif que lorsque le curseur de la souris se trouve sur une zone *FLpanel*.

## Exemples

Voici un exemple de l'opcode FLmouse. Il utilise le fichier *FLmouse.csd* [exemples/FLmouse.csd].

### Exemple 167. Exemple de l'opcode FLmouse.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```

```
-odac          -iadc      -d          ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera 2007
giwidth = 400
giheight = 300
FLpanel "FLmouse", giwidth, giheight, 10, 10
FLpanelEnd

FLrun

0dbfs = 1

instr 1
  kx, ky, kb1, kb2, kb3      FLmouse 2
  ktrig changed kx, ky  ;Print only if coordinates have changed
  printf "kx = %f   ky = %f \n", ktrig, kx, ky
  kfreq = ((giwidth - ky)*1000/giwidth) + 300

  ; y coordinate determines frequency, x coordinate determines amplitude
  ; Left mouse button (kb1) doubles the frequency
  ; Right mouse button (kb3) activates sound on channel 2
  aout oscil kx /giwidth , kfreq * (kb1 + 1), 1
  outs aout, aout * kb3
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1

i 1 0 120
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# flooper

flooper — Lecture en boucle basée sur une table de fonction avec fondu enchainé.

## Description

Cet opcode lit les données audio d'une table de fonction et les restitue dans une boucle dont le début, la durée et l'étendue du fondu enchainé sont fixés par l'utilisateur. On peut également contrôler la hauteur de la boucle ainsi que sa lecture à l'envers. Il accepte des tables dont la longueur n'est pas une puissance de deux, telles que celles de GEN01 avec allocation différée.

## Syntaxe

```
asig flooper kamp, kpitch, istart, idur, ifad, ifn
```

## Initialisation

*istart* -- début de la boucle en secondes

*idur* -- durée de la boucle en secondes

*ifad* -- étendue du fondu enchainé en secondes

*ifn* -- numéro de la table de fonction, généralement créée au moyen de GEN01

## Exécution

*asig* -- signal de sortie

*kon* -- contrôle de l'amplitude

*kpitch* -- contrôle de la hauteur (rapport de transposition) ; avec des valeurs négatives, la boucle est lue à l'envers.

## Exemples

### Exemple 168. Exemple

```
aout flooper 16000, 1, 1, 4, 0.05, 1 ; loop starts at 1 sec, for 4 secs, 0.05 crossfade  
out      aout
```

L'exemple ci-dessus montre l'opération de base de *flooper*. La hauteur peut être contrôlée au taux-k ainsi que l'amplitude. L'exemple suppose que la table 1 contient au moins 5.05 secondes de données audio (boucle durant 4 secondes, commençant 1 seconde après le début de la table, avec un fondu enchainé de 0.05 secondes après la fin de la boucle).

## Crédits



Auteur : Victor Lazzarini  
Avril 2005

Nouveau plugin dans la version 5

Avril 2005.

# flooper2

flooper2 — Lecture en boucle basée sur une table de fonction avec fondu enchainé.

## Description

Cet opcode implémente une lecture de boucle avec fondu enchainé, avec des paramètres variables, trois modes de boucle, et une utilisation facultative d'une table pour la forme du fondu enchainé. Il accepte comme source audio des tables dont la longueur n'est pas une puissance de deux, telles que celles de GEN01 avec allocation différée.

## Syntaxe

```
asig flooper2 kamp, kpitch, kloopstart, kloopend, kcrossfade, ifn \  
    [, istart, imode, ifenv, iskip]
```

## Initialisation

*ifn* -- numéro de la table de fonction de la source audio, généralement créée au moyen de GEN01

*istart* -- début de la lecture en secondes

*imode* -- modes de boucle : 0 à l'endroit, 1 à l'envers, 2 à l'endroit et à l'envers. (0 par défaut)

*ifenv* -- s'il est non nul, numéro de la table de la forme de l'enveloppe du fondu enchainé. La valeur par défaut, 0, donne un fondu enchainé linéaire.

*iskip* -- s'il vaut 1, l'initialisation de l'opcode est ignorée, pour les notes liées, l'exécution continuant depuis la position dans la boucle où la note précédente s'est terminée. Avec la valeur par défaut, 0, l'initialisation a lieu.

## Exécution

*asig* -- signal de sortie

*kamp* -- contrôle de l'amplitude

*kpitch* -- contrôle de la hauteur (rapport de transposition) ; les valeurs négatives sont interdites.

*kloopstart* -- début de la boucle (en secondes). Noter que bien qu'étant de taux-k, les paramètres de boucle comme celui-ci ne sont mis à jour qu'une fois par itération de la boucle.

*kloopend* -- fin de la boucle (en secondes), mis à jour une seule fois par itération de la boucle.

*kcrossfade* -- longueur du fondu enchainé (en secondes), mis à jour une seule fois par itération de la boucle et limité par la longueur de la boucle.

Avec le mode 1 de *imode* la boucle ne se fait qu'à l'envers depuis la fin jusqu'au début.

## Exemples

## Exemple 169. Exemple

```
aout flooper2 16000, 1, 1, 5, 0.05, 1 ; loop starts at 1 sec, for 4 secs, 0.05 crossfade  
out aout
```

L'exemple ci-dessus montre l'opération de base de *flooper2*. La hauteur peut être contrôlée au taux-k ainsi que l'amplitude et les paramètres de boucle. L'exemple suppose que la table 1 contient au moins 5.05 secondes de données audio (boucle durant 4 secondes, commençant 1 seconde après le début de la table, avec un fondu enchaîné de 0.05 secondes après la fin de la boucle). La boucle est en mode 0 (boucle normale à l'endroit).

## Crédits

Auteur : Victor Lazzarini  
Juillet 2006

Nouveau plugin dans la version 5

Juillet 2006.

# floor

floor — Retourne le plus grand entier inférieur ou égal à  $x$ .

## Description

Retourne le plus grand entier inférieur ou égal à  $x$ .

## Syntaxe

`floor(x)` (argument au taux d'initialisation, de contrôle ou audio)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Voir Aussi

*abs, exp, int, log, log10, i, sqrt*

## Crédits

Auteur : Istvan Varga  
Nouveau dans Csound 5  
2005

# FLpack

FLpack — Permet de concentrer et d'aligner des widgets FLTK.

## Description

*FLpack* permet de concentrer et d'aligner des widgets.

## Syntaxe

**FLpack** *iwidth, iheight, ix, iy, itype, ispace, iborder*

## Initialisation

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*itype* -- un nombre entier modifiant l'apparence du widget cible.

L'argument *itype* exprime le type de concentration :

- 0 - verticale
- 1 - horizontale

*ispace* -- fixe l'espace entre les widgets.

*iborder* -- type de la bordure du conteneur. Il est exprimé par un nombre entier choisi parmi les suivants :

- 0 - pas de bordure
- 1 - bordure de boîte en creux
- 2 - bordure de boîte saillante
- 3 - bordure gravée
- 4 - bordure en relief
- 5 - bordure ligne noire
- 6 - mince bordure en creux
- 7 - mince bordure saillante

## Exécution

*FLpack* permet de concentrer et d'aligner des widgets.

Les conteneurs sont utiles pour formater l'apparence graphique des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valeurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Exemples

L'exemple suivant :

```

FLpanel "Panel1", 450, 300, 100, 100
FLpack 400, 300, 10, 40, 0, 15, 3
gk1, ihs1      FLslider "FLslider 1", 500, 1000, 2, 1, -1, 300, 15, 20, 50
gk2, ihs2      FLslider "FLslider 2", 300, 5000, 2, 3, -1, 300, 15, 20, 100
gk3, ihs3      FLslider "FLslider 3", 350, 1000, 2, 5, -1, 300, 15, 20, 150
gk4, ihs4      FLslider "FLslider 4", 250, 5000, 1, 11, -1, 300, 30, 20, 200
gk5, ihs5      FLslider "FLslider 5", 220, 8000, 2, 1, -1, 300, 15, 20, 250
gk6, ihs6      FLslider "FLslider 6", 1, 5000, 1, 13, -1, 300, 15, 20, 300
gk7, ihs7      FLslider "FLslider 7", 870, 5000, 1, 15, -1, 300, 30, 20, 350
FLpackEnd
FLpanelEnd

```

...produira ce résultat, lorsque l'on changera la taille de la fenêtre :



FLpack.

## Voir Aussi

*FLgroup, FLgroupEnd, FLpackEnd, FLpanel, FLpanelEnd, FLscroll, FLscrollEnd, FLtabs, FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLpackEnd

FLpackEnd — Marque la fin d'un groupe de widgets FLTK concentrés ou alignés.

## Description

Marque la fin d'un groupe de widgets FLTK concentrés ou alignés.

## Syntaxe

`FLpackEnd`

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir Aussi

*FLgroup*, *FLgroupEnd*, *FLpack*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLpack\_end

FLpack\_End — Marque la fin d'un groupe de widgets FLTK concentrés ou alignés.

## Description

Marque la fin d'un groupe de widgets FLTK concentrés ou alignés. C'est un autre nom pour **FLpackEnd** fourni pour des raisons de compatibilité. Voir *FLpackEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLpanel

FLpanel — Crée une fenêtre contenant des widgets FLTK.

## Description

Crée une fenêtre contenant des widgets FLTK.

## Syntaxe

```
FLpanel "label", iwidth, iheight [, ix] [, iy] [, iborder] [, ikbdcapture] [, iclose]
```

## Initialisation

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* (facultatif) -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* (facultatif) -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iborder* (facultatif) -- type de la bordure du conteneur. Il est exprimé par un nombre entier choisi parmi les suivants :

- 0 - pas de bordure
- 1 - bordure de boîte en creux
- 2 - bordure de boîte saillante
- 3 - bordure gravée
- 4 - bordure en relief
- 5 - bordure ligne noire
- 6 - mince bordure en creux
- 7 - mince bordure saillante

*ikbdcapture* (0 par défaut) -- Si cet indicateur est positionné à 1, les événements du clavier sont capturés par la fenêtre (pour une utilisation par *sensekey* et par *FLkeyIn*)

*iclose* (0 par défaut) -- Si cet indicateur contient une valeur différente de 0, le bouton de fermeture de la fenêtre est désactivé, et la fenêtre ne peut pas être fermée directement par l'utilisateur. Elle se fermera à la sortie de Csound.

## Exécution

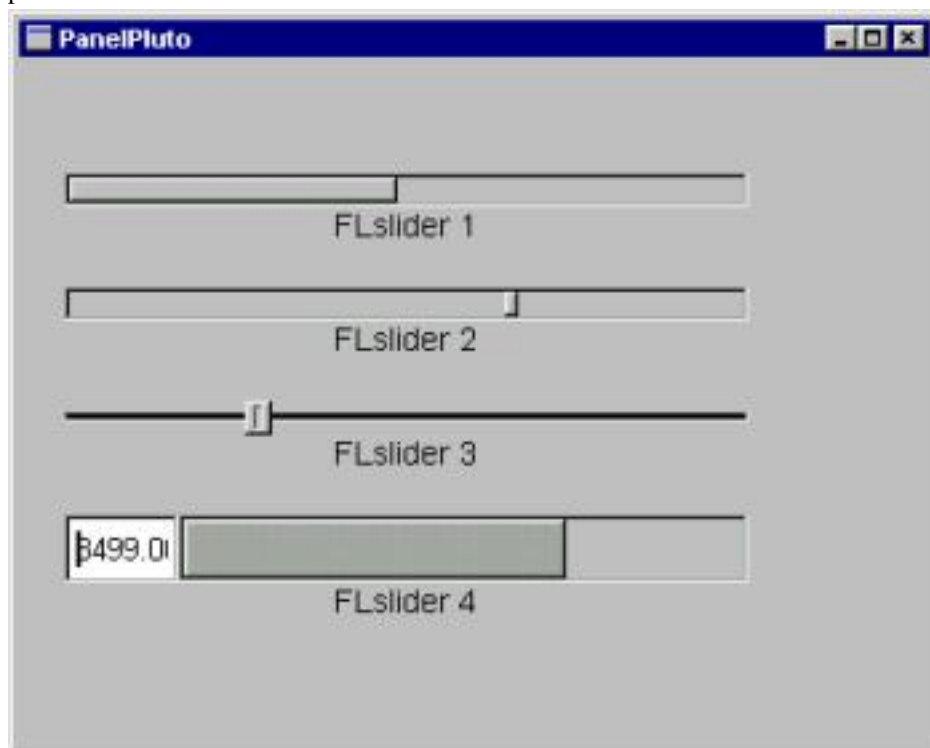
Les conteneurs sont utiles pour formater l'apparence graphique des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valeurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

*FLpanel* crée une fenêtre. Il doit être suivi d'un opcode *FLpanelEnd* après que tous les widgets qu'il contient aient été déclarés. Par exemple :

```
gk1, ih1 FLpanel "PanelPluto", 450, 550, 100, 100 ;***** start of container
gk2, ih2 FLslider "FLslider 1", 500, 1000, 2, 1, -1, 300, 15, 20, 50
gk3, ih3 FLslider "FLslider 2", 300, 5000, 2, 3, -1, 300, 15, 20, 100
gk4, ih4 FLslider "FLslider 3", 350, 1000, 2, 5, -1, 300, 15, 20, 150
FLpanelEnd ;***** end of container
```

produira le résultat suivant :



*FLpanel*.

Si l'indicateur *ikbdcapture* est positionné, la fenêtre capture les événements du clavier et les envoie à *sensekey*. Cet indicateur modifie le comportement de *sensekey*, et lui fait recevoir les événements depuis la fenêtre FLTK au lieu de stdin.

## Exemples

Voici un exemple de l'opcode *FLpanel*. Il utilise le fichier *FLpanel.csd* [examples/FLpanel.csd].

## Exemple 170. Exemple de l'opcode FLpanel.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLpanel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Creates an empty window panel
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

; Panel height in pixels
ipanelheight = 900
; Panel width in pixels
ipanelwidth = 400
; Horizontal position of the panel on screen in pixels
ix = 50
; Vertical position of the panel on screen in pixels
iy = 50

FLpanel "A Window Panel", ipanelheight, ipanelwidth, ix, iy
; End of panel contents
FLpanelEnd

;Run the widget thread!
FLrun

instr 1
endin

</CsInstruments>
<CsScore>

; 'Dummy' score event of 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLgroup, FLgroupEnd, FLpack, FLpackEnd, FLpanelEnd, FLscroll, FLscrollEnd, FLtabs, FLtabsEnd, sensekey*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLpanelEnd

FLpanelEnd — Marque la fin d'un groupe de widgets FLTK contenus dans une fenêtre (panel).

## Description

Marque la fin d'un groupe de widgets FLTK contenus dans une fenêtre (panel).

## Syntaxe

`FLpanelEnd`

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir Aussi

*FLgroup*, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLpanel\_end

FLpanel\_end — Marque la fin d'un groupe de widgets FLTK contenus dans une fenêtre (panel).

## Description

Marque la fin d'un groupe de widgets FLTK contenus dans une fenêtre (panel). C'est un autre nom pour **FLpanelEnd** fourni pour des raisons de compatibilité. Voir *FLpanelEnd*.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLprintk

FLprintk — Un opcode FLTK qui imprime une valeur de taux-k à intervalles donnés.

## Description

*FLprintk* est semblable à *printk* mais il montre les valeurs d'un signal de taux-k dans un champ texte plutôt que dans la console.

## Syntaxe

```
FLprintk itime, kval, idisp
```

## Initialisation

*itime* -- l'intervalle de temps en secondes entre deux mises à jour de l'affichage.

*idisp* -- la valeur d'un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut fixer ce paramètre à une valeur négative.

## Exécution

*kval* -- signal de taux-k à afficher.

*FLprintk* est semblable à *printk*, mais il montre les valeurs d'un signal de taux-k dans un champ texte plutôt que dans la console. L'argument *idisp* doit contenir la valeur du *ihandle* retourné par un opcode *FLvalue* précédent. Alors que *FLvalue* doit être placé dans la section d'en-tête d'un orchestre dans un bloc *FLpanel/FLpanelEnd*, *FLprintk* doit être placé dans un instrument pour opérer correctement. Pour cette raison, il ralentit l'exécution et il ne faut l'utiliser que pour des raisons de débogages.

## Voir Aussi

*FLbox*, *FLbutBank*, *FLbutton*, *FLprintk2*, *FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLprintk2

FLprintk2 — Un opcode FLTK qui imprime une nouvelle valeur chaque fois qu'une variable au taux-k change.

## Description

*FLprintk2* est semblable à *FLprintk* mais il ne montre la valeur d'une variable de taux-k que lorsqu'elle change.

## Syntaxe

```
FLprintk2 kval, idisp
```

## Initialisation

*idisp* -- la valeur d'un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut fixer ce paramètre à une valeur négative.

## Exécution

*kval* -- signal de taux-k à afficher.

*FLprintk2* est semblable à *FLprintk*, mais il ne montre la valeur d'une variable de taux-k que lorsqu'elle change. Utile pour surveiller les changements de contrôle MIDI lorsque l'on utilise des réglettes. Il ne faut l'utiliser que pour des raisons de débogages, car il ralentit l'exécution.

## Voir Aussi

*FLbox*, *FLbutBank*, *FLbutton*, *FLprintk*, *FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLroller

FLroller — Un widget FLTK qui crée une molette.

## Description

*FLroller* est une sorte de bouton rotatif, mais disposé transversalement (une molette).

## Syntaxe

```
kout, ihandle FLroller "label", imin, imax, istep, iexp, itype, idisp, \
    iwidth, iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLroller* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- mvaleur minimale de l'intervalle de sortie.

*imax* -- valeur maximale de l'intervalle de sortie.

*istep* -- un nombre en virgule flottante indiquant le pas d'incréméntation du valuateur à chaque clic de souris. L'argument *istep* permet de ralentir le mouvement de la molette, ce qui autorise une précision arbitraire.

*iexp* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexp* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.



### IMPORTANT !

Noter que les tables utilisées par les valuateurs doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En fait, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

*itype* -- un nombre entier indiquant l'apparence du valuateur.

L'argument *itype* accepte les valeurs suivantes :

- 1 - molette horizontale
- 2 - molette verticale

*idisp* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*kout* -- valeur en sortie.

*FLroller* est une sorte de bouton rotatif, mais disposé transversalement (une molette) :



FLroller.

## Exemples

Voici un exemple de l'opcode *FLroller*. Il utilise le fichier *FLroller.csd* [examples/FLroller.csd].

### Exemple 171. Exemple de l'opcode *FLroller*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLroller.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flroller controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Roller", 900, 400, 50, 50
; Minimum value output by the roller
imin = 200
; Maximum value output by the roller
imax = 5000
; Increment with each pixel
```

```

        istep = 1
        ; Logarithmic type roller selected
        iexp = -1
        ; Roller graphic type (1=horizontal)
        itype = 1
        ; Display handle (-1=not used)
        idisp = -1
        ; Width of the roller in pixels
        iwidth = 300
        ; Height of the roller in pixels
        iheight = 50
        ; Distance of the left edge of the knob
        ; from the left edge of the panel
        ix = 300
        ; Distance of the top edge of the knob
        ; from the top edge of the panel
        iy = 50

        gkfreq, ihandle FLroller "Frequency", imin, imax, istep, iexp, itype, idisp, iwidth, iheight, ix, iy
        ; End of panel contents
FLpanelEnd
        ; Run the widget thread!
FLrun

instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, gkfreq, ifn
    out asig
endin

</CsInstruments>
<CsScore>

    ; Function table that defines a single cycle
    ; of a sine wave.
f 1 0 1024 10 1

    ; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLcount, FLjoy, FLknob, FLslider, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLrun

FLrun — Démarre le processus léger des widgets FLTK.

## Description

Démarre le processus léger des widgets FLTK.

## Syntaxe

`FLrun`

## Exécution

Cet opcode doit être placé à la fin des déclaration de tous les widgets. Il n'a pas d'arguments et il sert à démarrer le processus léger relatif aux widgets. Les widgets ne seront pas opérationnels si *FLrun* est absent.

## Voir Aussi

*FLgetsnap, FLloadsnap, FLsavesnap, FLsetsnap, FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsavesnap

FLsavesnap — Sauvegarde dans un fichier tous les instantanés actuellement créés.

## Description

*FLsavesnap* sauvegarde dans un fichier tous les instantanés actuellement créés (c'est-à-dire la banque de mémoire en entier).

## Syntaxe

```
FLsavesnap "filename" [, igroup]
```

## Initialisation

« *filename* » -- une chaîne de caractères entre guillemets correspondant à un fichier dans lequel sauvegarder une banque d'instantanés.

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valueurs. La valeur par défaut est zéro qui fait référence au premier groupe. Le numéro de groupe est déterminé par l'opcode *FLsetSnapGroup*.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

*FLsavesnap* sauvegarde tous les instantanés actuellement créés (c'est-à-dire la banque de mémoire en entier) dans un fichier dont le nom est *filename*. Comme le fichier est un fichier texte, les valeurs des instantanés peuvent être modifiées manuellement dans un éditeur de texte. Le format des données du fichier est le suivant (pour le moment, ceci pouvant changer dans une prochaine version de Csound) :

```
----- 0 -----
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLslider 331.946 80 5000 -1 "frequency of the first oscillator"
FLslider 385.923 80 5000 -1 "frequency of the second oscillator"
FLslider 80 80 5000 -1 "frequency of the third oscillator"
FLcount 0 0 10 0 "this index must point to the location number where snapshot is stored"
FLbutton 0 0 1 0 "Store snapshot to current index"
FLbutton 0 0 1 0 "Save snapshot bank to disk"
FLbutton 0 0 1 0 "Load snapshot bank from disk"
FLbox 0 0 1 0 ""
----- 1 -----
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLslider 819.72 80 5000 -1 "frequency of the first oscillator"
FLslider 385.923 80 5000 -1 "frequency of the second oscillator"
FLslider 80 80 5000 -1 "frequency of the third oscillator"
FLcount 1 0 10 0 "this index must point to the location number where snapshot is stored"
FLbutton 0 0 1 0 "Store snapshot to current index"
```

```
FLbutton 0 0 1 0 "Save snapshot bank to disk"
FLbutton 0 0 1 0 "Load snapshot bank from disk"
FLbox 0 0 1 0 ""
----- 2 -----
..... etc...
----- 3 -----
..... etc...
-----
```

Comme on peut le voir, chaque instantané contient plusieurs lignes. Chaque instantané est séparé du précédent et du suivant par une ligne de cette sorte :

"----- Num d'instantané -----"

Suivent plusieurs lignes contenant les données. Chacune de ces lignes correspond à un widget.

Le premier champ de chaque ligne est une chaîne de caractères sans guillemets contenant le nom de l'opcode correspondant au widget. Le second champ est un nombre exprimant la valeur courante d'un instantané. Dans la version actuelle, c'est le seul champ que l'on peut modifier manuellement. Les troisième et quatrième champs montrent les valeurs minimale et maximale pour ce valuateur. Le cinquième champ est un nombre spécial qui indique si le valuateur est linéaire (valeur 0), exponentiel (valeur -1), ou est indexé par une table avec interpolation des valeurs (numéro de table négatif) ou sans interpolation (numéro de table positif). Le dernier champ, une chaîne de caractères entre guillemets, contient l'étiquette du widget. La dernière ligne du fichier est toujours

"-----"

.

Noter que *FLvalue* et *FLbox* ne sont pas des valuateurs et que leurs valeurs sont constantes, ne pouvant pas être modifiées.

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

## Exemples

Voici un exemple simple de sauvegarde d'un instantané FLTK. Il utilise le fichier *FLsavesnap\_simple.csd* [examples/FLsavesnap\_simple.csd].

### Exemple 172. Exemple de sauvegarde d'un instantané FLTK.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>
```

```

sr=48000
ksmps=128
nchnls=2

; Example by Hector Centeno and Andres Cabrera 2007

; giSWMTab4 ftgen 0, 0, 513, 21, 10, 1, .3
; giSWMTab4M ftgen 0, 0, 64, 7, 1, 50, 1

FLpanel "Snapshots", 530, 190, 40, 410, 3
  FLcolor 100, 118, 140
  ivalSM1          FLvalue "", 70, 20, 270, 20
  gksliderA, gislidSM1      FLslider "Slider", -4, 4, 0, 3, ivalSM1, 250, 20, 20, 20
  itext1          FLbox "store", 1, 1, 14, 50, 25, 355, 15
  itext2          FLbox "load", 1, 1, 14, 50, 25, 415, 15
  gksnap, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 364, 45, 0, 3, 0, 3, 1
  gksnap, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 364, 75, 0, 3, 0, 3, 2
  gksnap, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 364, 105, 0, 3, 0, 3, 3
  gksnap, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 364, 135, 0, 3, 0, 3, 4

  gkload, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 424, 45, 0, 4, 0, 3, 1
  gkload, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 424, 75, 0, 4, 0, 3, 2
  gkload, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 424, 105, 0, 4, 0, 3, 3
  gkload, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 424, 135, 0, 4, 0, 3, 4

  ivalSM2          FLvalue "", 70, 20, 270, 80
  gkknobA, gislidSM2      FLknob "Knob", -4, 4, 0, 3, ivalSM2, 60, 120, 60
FLpanelEnd
FLsetVal_i 1, gislidSM1
FLsetVal_i 1, gislidSM2
FLrun

instr 1
  endin

instr 3 ; Save snapshot
index init 0
ipstno = p4
Sfile sprintf "snapshot_simple.%d.snap", ipstno

inumsnap, inumval FLsetsnap index ;, -1, igroup
FLsavesnap Sfile

  endin

instr 4 ;Load snapshot
index init 0
ipstno = p4
Sfile sprintf "snapshot_simple.%d.snap", ipstno

FLloadsnap Sfile
inumload FLgetsnap index ;, igroup

  endin

</CsInstruments>
<CsScore>
f 0 3600

e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de sauvegarde d'instantané FLTK utilisant des groupes d'instantanés. Il utilise le fichier *FLsavesnap.csd* [exemples/FLsavesnap.csd].

**Exemple 173.** Exemple de sauvegarde d'instantané FLTK utilisant des groupes d'instantanés.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=48000
ksmps=128
nchnls=2

; Example by Hector Centeno and Andres Cabrera 2007

; giSWMtab4 ftgen 0, 0, 513, 21, 10, 1, .3
; giSWMtab4M ftgen 0, 0, 64, 7, 1, 50, 1

FLpanel "Snapshots", 530, 350, 40, 410, 3
FLcolor 100, 118, 140
FLsetSnapGroup 0
ivalSM1          FLvalue  "", 70, 20, 270, 20
ivalSM2          FLvalue  "", 70, 20, 270, 60
ivalSM3          FLvalue  "", 70, 20, 270, 100
ivalSM4          FLvalue  "", 70, 20, 270, 140
gksliderA, gislidSM1  FLslider "Slider A", -4, 4, 0, 3, ivalSM1, 250, 20, 20, 20
gksliderB, gislidSM2  FLslider "Slider B", 1, 10, 0, 3, ivalSM2, 250, 20, 20, 60
gksliderC, gislidSM3  FLslider "Slider C", 0, 1, 0, 3, ivalSM3, 250, 20, 20, 100
gksliderD, gislidSM4  FLslider "Slider D", 0, 1, 0, 3, ivalSM4, 250, 20, 20, 140
itext1          FLbox  "store", 1, 1, 14, 50, 25, 355, 15
itext2          FLbox  "load", 1, 1, 14, 50, 25, 415, 15
itext3          FLbox  "G\nr\no\nu\np\n\n\n", 1, 1, 14, 30, 145, 485, 15
gksnap, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 364, 45, 0, 3, 0, 3, 1
gksnap, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 364, 75, 0, 3, 0, 3, 2
gksnap, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 364, 105, 0, 3, 0, 3, 3
gksnap, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 364, 135, 0, 3, 0, 3, 4
gkload, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 424, 45, 0, 4, 0, 3, 1
gkload, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 424, 75, 0, 4, 0, 3, 2
gkload, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 424, 105, 0, 4, 0, 3, 3
gkload, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 424, 135, 0, 4, 0, 3, 4

FLcolor 100, 140, 118
FLsetSnapGroup 1
ivalSM5          FLvalue  "", 70, 20, 270, 190
ivalSM6          FLvalue  "", 70, 20, 270, 230
ivalSM7          FLvalue  "", 70, 20, 270, 270
ivalSM8          FLvalue  "", 70, 20, 270, 310
gkknobA, gislidSM5  FLknob  "Knob A", -4, 4, 0, 3, ivalSM5, 45, 10, 230
gkknobB, gislidSM6  FLknob  "Knob B", 1, 10, 0, 3, ivalSM6, 45, 75, 230
gkknobC, gislidSM7  FLknob  "Knob C", 0, 1, 0, 3, ivalSM7, 45, 140, 230
gkknobD, gislidSM8  FLknob  "Knob D", 0, 1, 0, 3, ivalSM8, 45, 205, 230
itext4          FLbox  "store", 1, 1, 14, 50, 25, 355, 185
itext5          FLbox  "load", 1, 1, 14, 50, 25, 415, 185
itext6          FLbox  "G\nr\no\nu\np\n\n\n", 1, 1, 14, 30, 145, 485, 185
gksnap, ibuttn1  FLbutton "5", 1, 0, 11, 25, 25, 364, 215, 0, 3, 0, 3, 5
gksnap, ibuttn2  FLbutton "6", 1, 0, 11, 25, 25, 364, 245, 0, 3, 0, 3, 6
gksnap, ibuttn3  FLbutton "7", 1, 0, 11, 25, 25, 364, 275, 0, 3, 0, 3, 7
gksnap, ibuttn4  FLbutton "8", 1, 0, 11, 25, 25, 364, 305, 0, 3, 0, 3, 8
gkload, ibuttn1  FLbutton "5", 1, 0, 11, 25, 25, 424, 215, 0, 4, 0, 3, 5
gkload, ibuttn2  FLbutton "6", 1, 0, 11, 25, 25, 424, 245, 0, 4, 0, 3, 6
gkload, ibuttn3  FLbutton "7", 1, 0, 11, 25, 25, 424, 275, 0, 4, 0, 3, 7
gkload, ibuttn4  FLbutton "8", 1, 0, 11, 25, 25, 424, 305, 0, 4, 0, 3, 8

FLpanelEnd
FLsetVal_i 1, gislidSM1
FLsetVal_i 1, gislidSM2
FLsetVal_i 0, gislidSM3
FLsetVal_i 0, gislidSM4
FLsetVal_i 1, gislidSM5
FLsetVal_i 1, gislidSM6
FLsetVal_i 0, gislidSM7
FLsetVal_i 0, gislidSM8
FLrun

instr 1

endin
```



```

instr 3 ; Save snapshot
index init 0
ipstno = p4
igroup = 0
Sfile sprintf "PVCsynth.%d.snap", ipstno
if ipstno > 4 then
    igroup = 1
endif

    inumsnap, inumval FLsetsnap index , -1, igroup
FLsavesnap Sfile

    endin

instr 4 ;Load snapshot
index init 0
ipstno = p4
igroup = 0
Sfile sprintf "PVCsynth.%d.snap", ipstno
if ipstno > 4 then
    igroup = 1
endif

FLloadsnap Sfile
    inumload FLgetsnap index , igroup

    endin

</CsInstruments>
<CsScore>
    ;Dummy table for FLgetsnap
    ; f 1 0 1024 10 1
    f 0 3600

e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLgetsnap, FLloadsnap, FLsetSnapGroup, FLrun, FLsetsnap, FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLscroll

FLscroll — Un opcode FLTK qui ajoute des barres d'ascenseur à une zone.

## Description

*FLscroll* ajoute des barres d'ascenseur à une zone.

## Syntaxe

```
FLscroll iwidth, iheight [, ix] [, iy]
```

## Initialisation

*iwidth* -- largeur widget.

*iheight* -- hauteur du widget.

*ix* (facultatif) -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* (facultatif) -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

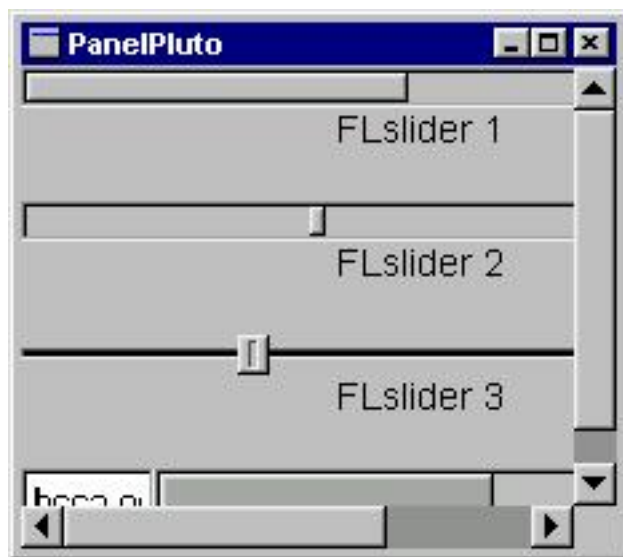
Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

*FLscroll* ajoute des barres d'ascenseur à une zone. Normalement il faut fixer les arguments *iwidth* et *iheight* à la même valeur que ceux de la fenêtre parente ou d'un autre conteneur parent. *ix* et *iy* sont facultatifs car ils sont normalement fixés à zéro. Par exemple le code suivant :

```
FLpanel      "PanelPluto", 400, 300, 100, 100
FLscroll    400, 300
gk1, ih1 FLslider "FLslider 1", 500, 1000, 2, 1, -1, 300, 15, 20, 50
gk2, ih2 FLslider "FLslider 2", 300, 5000, 2, 3, -1, 300, 15, 20, 100
gk3, ih3 FLslider "FLslider 3", 350, 1000, 2, 5, -1, 300, 15, 20, 150
gk4, ih4 FLslider "FLslider 4", 250, 5000, 1, 11, -1, 300, 30, 20, 200
FLscrollEnd
FLpanelEnd
```

montrera des barres d'ascenseur quand la taille de la fenêtre principale sera diminuée.



FLscroll.

## Exemples

Voici un exemple de l'opcode FLscroll. Il utilise le fichier *FLscroll.csd* [examples/FLscroll.csd].

### Exemple 174. Exemple de l'opcode FLscroll.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLscroll.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flscroll opcode which enables
; the use of widget sizes and placings beyond the
; dimensions of the containing panel
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Text Box", 420, 200, 50, 50
  iwidth = 420
  iheight = 200
  ix = 0
  iy = 0
  FLscroll iwidth, iheight, ix, iy
  ih3 FLbox "DRAG THE SCROLL BAR TO THE RIGHT IN ORDER TO READ THE REST OF THIS TEXT!", 1, 10, 20, 87
  FLscrollEnd
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
endin
```

```
</CsInstruments>
<CsScore>

; 'Dummy' score event of 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLgroup, FLgroupEnd, FLpack, FLpackEnd, FLpanel, FLpanelEnd, FLscrollEnd, FLtabs, FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLscrollEnd

FLscrollEnd — Un opcode FLTK qui marque la fin d'une zone avec barres d'ascenseur.

## Description

Un opcode FLTK qui marque la fin d'une zone avec barres d'ascenseur.

## Syntaxe

**FLscrollEnd**

Exécution

## Performance

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valeurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir Aussi

*FLgroup*, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLtabs*, *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLscroll\_end

FLscroll\_end — Un opcode FLTK qui marque la fin d'une zone avec barres d'ascenseur.

## Description

Un opcode FLTK qui marque la fin d'une zone avec barres d'ascenseur. C'est un autre nom pour **FLscrollEnd** fourni pour des raisons de compatibilité. Voir *FLscrollEnd*.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetAlign

FLsetAlign — Fixe l'alignement du texte de l'étiquette d'un widget FLTK.

## Description

*FLsetAlign* fixe l'alignement du texte de l'étiquette d'un widget FLTK.

## Syntaxe

**FLsetAlign** *ialign*, *ihandle*

## Initialisation

*ialign* -- fixe l'alignement du texte de l'étiquette d'un widget.

Les valeurs admises pour l'argument *ialign* sont :

- 1 - centré
- 2 - en haut
- 3 - en bas
- 4 - à gauche
- 5 - à droite
- 6 - en haut à gauche
- 7 - en haut à droite
- 8 - en bas à gauche
- 9 - en bas à droite

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetBox

FLsetBox — Fixe l'apparence d'une boîte entourant un widget FLTK.

## Description

*FLsetBox* fixe l'apparence d'une boîte entourant le widget cible.

## Syntaxe

**FLsetBox** *itype*, *ihandle*

## Initialisation

*itype* -- un nombre entier qui modifie l'apparence du widget cible.

Les valeurs admises pour l'argument *itype* sont :

- 1 - boîte sans relief
- 2 - boîte saillante
- 3 - boîte en creux
- 4 - boîte légèrement saillante
- 5 - boîte légèrement en creux
- 6 - boîte gravée
- 7 - boîte en relief
- 8 - boîte avec cadre
- 9 - boîte ombrée
- 10 - boîte arrondie
- 11 - boîte arrondie ombrée
- 12 - boîte arrondie sans relief
- 13 - boîte arrondie saillante
- 14 - boîte arrondie creuse
- 15 - boîte en losange saillante
- 16 - boîte en losange en creux
- 17 - boîte ovale
- 18 - boîte ovale ombrée



- 19 - boîte ovale sans relief

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetColor

FLsetColor — Fixe la couleur d'un widget FLTK.

## Description

*FLsetColor* fixe la couleur du widget cible.

## Syntaxe

**FLsetColor** *ired, igreen, iblue, ihandle*

## Initialisation

*ired* -- La composante rouge de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Exemples

Voici un exemple de l'opcode FLsetcolor. Il utilise le fichier *FLsetcolor.csd* [exemples/FLsetcolor.csd].

### Exemple 175. Exemple de l'opcode FLsetcolor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac       -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLsetcolor.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Using the opcode flsetcolor to change from the
; default colours for widgets
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Coloured Sliders", 900, 360, 50, 50
gkfreq, ihandle FLslider "A Red Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 50
```

```

ired1 = 255
igreen1 = 0
ibluel = 0
FLsetColor ired1, igreen1, ibluel, ihandle

gkfreq, ihandle FLslider "A Green Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 150
ired1 = 0
igreen1 = 255
ibluel = 0
FLsetColor ired1, igreen1, ibluel, ihandle

gkfreq, ihandle FLslider "A Blue Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 250
ired1 = 0
igreen1 = 0
ibluel = 255
FLsetColor ired1, igreen1, ibluel, ihandle
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
endin

</CsInstruments>
<CsScore>

; 'Dummy' score event for 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLsetColor2

FLsetColor2 — Fixe la couleur de sélection d'un widget FLTK.

## Description

*FLsetColor2* fixe la couleur de sélection du widget cible.

## Syntaxe

**FLsetColor2** *ired, igreen, iblue, ihandle*

## Initialisation

*ired* -- La composante rouge de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetFont

FLsetFont — Fixe le type de la police d'un widget FLTK.

## Description

*FLsetFont* fixe le type de la police du widget cible.

## Syntaxe

**FLsetFont** ifont, ihandle

## Initialisation

*ifont* -- fixe le type de la police de l'étiquette d'un widget.

Les valeurs admises pour l'argument *ifont* sont :

- 1 - helvetica (comme "Arial" sous Windows)
- 2 - helvetica gras
- 3 - helvetica italique
- 4 - helvetica gras italique
- 5 - courier
- 6 - courier gras
- 7 - courier italique
- 8 - courier gras italique
- 9 - times
- 10 - times gras
- 11 - times italique
- 12 - times gras italique
- 13 - symbol
- 14 - screen
- 15 - screen gras
- 16 - dingbats

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de

*ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetPosition

FLsetPosition — Fixe la position d'un widget FLTK.

## Description

*FLsetPosition* fixe la position du widget cible en fonction des arguments *ix* et *iy*.

## Syntaxe

```
FLsetPosition ix, iy, ihandle
```

## Initialisation

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetSize

FLsetSize — Redimensionne un widget FLTK.

## Description

*FLsetSize* redimensionne le widget cible (pas la taille de son texte) en fonction des arguments *iwidth* et *iheight*.

## Syntaxe

```
FLsetSize iwidth, iheight, ihandle
```

## Initialisation

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLsetsnap

FLsetsnap — Enregistre l'état courant de tous les valuateurs FLTK dans un instantané.

## Description

*FLsetsnap* enregistre l'état courant de tous les valuateurs présents dans l'orchestre dans un instantané (en mémoire).

## Syntaxe

```
inumsnap, inumval FLsetsnap index [, ifn, igroup]
```

## Initialisation

*inumsnap* -- nombre courant d'instantanés.

*inumval* -- nombre de valuateurs (dont la valeur est enregistrée dans l'instantané) présents dans l'orchestre courant.

*index* -- un nombre faisant référence de manière univoque à un instantané. Plusieurs instantanés peuvent être enregistrés dans la même banque.

*ifn* (optional) -- optional argument referring to an already allocated table, to store values of a snapshot.

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valuateurs. La valeur par défaut est zéro qui fait référence au premier groupe. Le numéro de groupe est déterminé par l'opcode *FLsetSnapGroup*.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

L'opcode *FLsetsnap* enregistre l'état courant de tous les valuateurs dans l'orchestre dans un instantané (en mémoire). On peut stocker n'importe quel nombre d'instantanés dans la banque courante. Les banques sont des structures qui n'existent qu'en mémoire, sans autre référence que le fait qu'on peut y accéder par les opcodes *FLsetsnap*, *FLsavesnap*, *FLloadsnap* et *FLgetsnap*. Il ne peut y avoir qu'une seule banque présente en mémoire.

Si l'argument facultatif *ifn* fait référence à une table valide déjà allouée, l'instantané sera enregistré dans la table plutôt que dans la banque. Ainsi cette table est accessible depuis d'autres opcodes de Csound.

L'argument *index* fait référence à un instantané déterminé de manière univoque. Si la valeur d'*index* fait référence à un instantané antérieurement sauvegardé, toutes ses anciennes valeurs seront remplacées par les valeurs courantes. Si *index* fait référence à un instantané qui n'existe pas, un nouvel instantané sera créé. Si la valeur d'*index* n'est pas adjacente à celle d'un instantané déjà créé, des instantanés vides seront créés. Par exemple, si la position d'*index* 0 contient le seul instantané présent dans une banque et que l'on sauvegarde un nouvel instantané avec l'*index* 5, toutes les positions entre 1 et 4 contiendront auto-

matiquement des instantanés vides. Les instantanés vides ne contiennent pas de données et sont neutres.

*FLsetsnap* retourne le nombre courant d'instantanés (l'argument *inumsnap*) et le nombre total de valeurs stockées dans chaque instantané (*inumval*). *inumval* est égal au nombre de valuateurs présents dans l'orchestre.

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

## Voir Aussi

*FLgetsnap*, *FLloadsnap*, *FLsetSnapGroup*, *FLrun*, *FLsavesnap*, *FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetSnapGroup

FLsetSnapGroup — Détermine le groupe d'instantané pour les valuateurs FLTK.

## Description

*FLsetSnapGroup* détermine le groupe d'instantané des valuateurs déclarés après lui.

## Syntaxe

**FLsetSnapGroup** *igroup*

## Initialisation

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valuateurs.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

*FLsetSnapGroup* détermine le groupe d'instantané d'un valuateur déclaré. Pour qu'un valuateur appartienne à un groupe fixé, il faut placer *FLsetSnapGroup* juste avant la déclaration du widget lui-même. Le groupe fixé par *FLsetSnapGroup* est valable pour tous les valuateurs déclarés après lui, jusqu'à ce qu'une nouvelle instruction *FLsetSnapGroup* avec un groupe différent soit rencontrée. Si aucune instruction *FLsetSnapGroup* n'est présente dans l'orchestre, le groupe par défaut pour tous les widgets sera le groupe zéro.

## Voir Aussi

*FLgetsnap*, *FLsetsnap*, *FLloadsnap*, *FLsavesnap*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetText

FLsetText — Fixe l'étiquette d'un widget FLTK.

## Description

*FLsetText* met la chaîne de caractères entre guillemets de l'argument *itext* dans l'étiquette du widget cible.

## Syntaxe

```
FLsetText "itext", ihandle
```

## Initialisation

« *itext* » -- une chaîne de caractères entre guillemets contenant le texte de l'étiquette du widget.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Exemples

Voici un exemple de l'opcode FLsetText. Il utilise le fichier *FLsetText.csd* [exemples/FLsetText.csd].

### Exemple 176. Exemple de l'opcode FLsetText.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLsetText.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

; Example by Giorgio Zucco and Andres Cabrera 2007

FLpanel "FLsetText",250,100,50,50

gkl,giha FLcount "", 1, 20, 1, 20, 1, 200, 40, 20, 20, 0, 1, 0, 1

FLpanelEnd
FLrun

instr 1
; This instrument is triggered by FLcount above each time
```

```
; its value changes
iname = i(gkl)
print iname
; Must use FLsetText on the init pass!
if (iname == 1) igoto text1
if (iname == 2) igoto text2
if (iname == 3) igoto text3

igoto end

text1:
FLsetText "FM",giha
igoto end

text2:
FLsetText "GRANUL",giha
igoto end

text3:
FLsetText "PLUCK",giha
igoto end

end:
    endin

</CsInstruments>
<CsScore>

f 0 3600

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetTextColor

FLsetTextColor — Fixe la couleur du texte de l'étiquette d'un widget FLTK.

## Description

*FLsetTextColor* fixe la couleur du texte de l'étiquette du widget cible.

## Syntaxe

**FLsetTextColor** *ired, iblue, igreen, ihandle*

## Initialisation

*ired* -- La composante rouge de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetTextSize

FLsetTextSize — Fixe la taille du texte de l'étiquette d'un widget FLTK.

## Description

*FLsetTextSize* fixe la taille du texte de l'étiquette du widget cible.

## Syntaxe

```
FLsetTextSize isize, ihandle
```

## Initialisation

*isize* -- taille de la police du widget cible. Les valeurs normales sont de l'ordre de 15. Des nombres plus grands augmentent la taille de la police, tandis que des nombres plus petits la réduisent.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetTextType

FLsetTextType — Fixe quelques attributs de la police du texte de l'étiquette d'un widget FLTK.

## Description

*FLsetTextType* fixe quelques attributs de la police du texte de l'étiquette du widget cible.

## Syntaxe

```
FLsetTextType itype, ihandle
```

## Initialisation

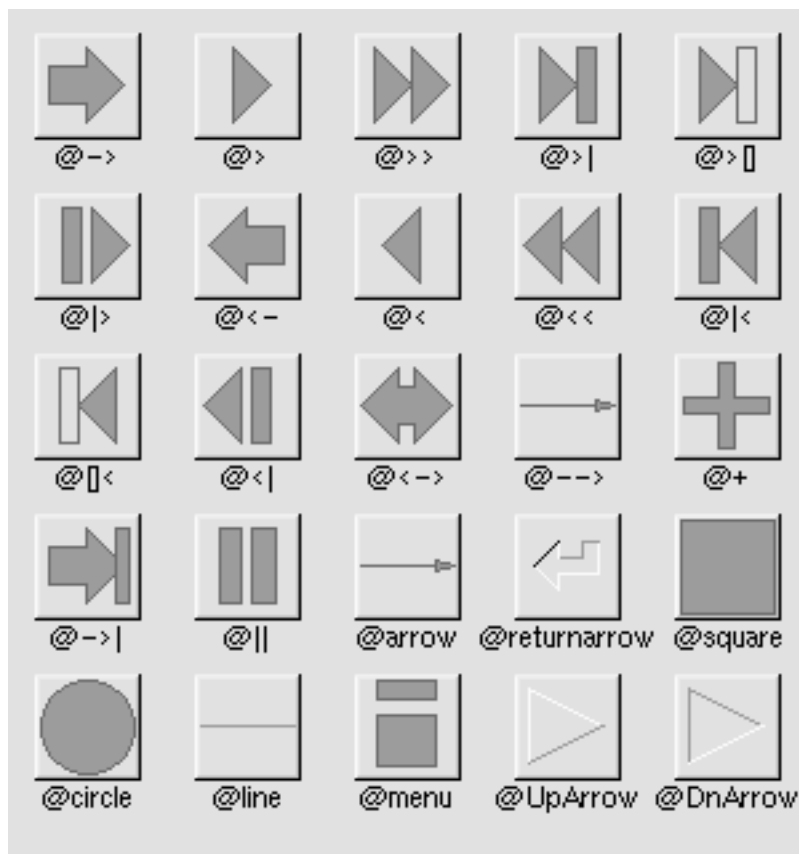
*itype* -- un nombre entier qui modifie l'apparence du widget cible.

Les valeurs admises pour l'argument *itype* sont :

- 0 - étiquette normale
- 1 - pas d'étiquette (le texte est caché)
- 2 - étiquette pictogramme (voir ci-dessous)
- 3 - étiquette ombrée
- 4 - étiquette gravée
- 5 - étiquette en relief
- 6 - étiquette bitmap (pas encore implémenté)
- 7 - étiquette pixmap (pas encore implémenté)
- 8 - étiquette image (pas encore implémenté)
- 9 - étiquette multiple (pas encore implémenté)
- 10 - étiquette de type libre (pas encore implémenté)

Lorsque l'on utilise *itype*=3 (étiquette pictogramme), il est possible d'affecter un symbole graphique à la place du texte de l'étiquette du widget cible. Dans ce cas, la chaîne de caractères de l'étiquette cible doit toujours commencer par un « @ ». Si elle commence avec un autre caractère (ou que le symbole n'est pas trouvé), l'étiquette est dessinée normalement. Les symboles suivants sont supportés :





Symboles d'étiquette FLTK supportés.

Le signe @ peut être suivi par les caractères de « formatage » facultatifs suivants, dans cet ordre :

1. « # »force une image carrée sans distortion de la forme du widget.
2. +[1-9] or -[1-9] grossit ou diminue l'image.
3. [1-9] effectue une rotation d'un multiple de 45 degrés. « 6 » ne fait rien, les autres valeurs pointent dans la direction de cette touche sur un pavé numérique.

Noter qu'avec *FLbox* et *FLbutton* il n'est pas nécessaire d'appeler l'opcode *FLsetTextType* pour utiliser un symbole. Dans ce cas, il suffit d'utiliser une étiquette commençant par « @ » suivi de la chaîne de formatage correcte.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetVal\_i

FLsetVal\_i — Met un nombre fourni par l'utilisateur dans la valeur d'un valuateur FLTK.

## Description

*FLsetVal\_i* force la valeur d'un valuateur à un nombre fourni par l'utilisateur.

## Syntaxe

**FLsetVal\_i** ivalue, ihandle

## Initialisation

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Exécution

*ivalue* -- Valeur à attribuer au widget.



### Note

*FLsetVal* n'est pas complètement implémenté, et il peut planter dans certains cas (par exemple en fixant la valeur d'un widget connecté au widget *FLvalue* - dans ce cas utiliser deux *FLsetVal\_i* séparés).

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetVal

FLsetVal — Fixe la valeur d'un valuateur FLTK au taux de contrôle.

## Description

*FLsetVal* est presque identique à *FLsetVal\_i*. Sauf qu'il opère au taux-k et qu'il n'affecte le valuateur cible que lorsque *ktrig* est fixé à une valeur différente de zéro.

## Syntaxe

```
FLsetVal ktrig, kvalue, ihandle
```

## Initialisation

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Exécution

*ktrig* -- déclenche l'opcode lorsqu'il est différent de 0.

*kvalue* -- Valeur à attribuer au widget.



### Note

*FLsetVal* n'est pas complètement implémenté, et il peut planter dans certains cas (par exemple en fixant la valeur d'un widget connecté au widget *FLvalue* - dans ce cas utiliser deux *FLsetVal* séparés).

## Voir Aussi

*FLcolor*, *FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLshow

FLshow — Rend visible un widget FLTK antérieurement caché.

## Description

*FLshow* rend visible un widget antérieurement caché.

## Syntaxe

**FLshow** *ihandle*

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*).

## Voir Aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLslidBnk

FLslidBnk — Un widget FLTK contenant un banc de réglettes horizontales.

## Description

*FLslidBnk* est un widget FLTK contenant un banc de réglettes horizontales.

## Syntaxe

```
FLslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \  
[, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]
```

## Initialisation

« *names* » -- une chaîne de caractères entre guillemets contenant le nom de chaque réglette. Chaque réglette peut avoir un nom différent. Chaque nom est séparé par un caractère « @ », par exemple : « fréquence@amplitude@coupure ». Il est possible de ne fournir aucun nom en donnant un seul espace « ». Dans ce cas, l'opcode affectera automatiquement un numéro en progression ascendante comme étiquette pour chaque réglette.

*inumsliders* -- le nombre de réglettes.

*ioutable* (facultatif, 0 par défaut) -- numéro d'une table allouée préalablement dans laquelle seront stockées les valeurs de sortie de chaque réglette. Il faut s'assurer que la taille de la table est suffisante pour contenir toutes les cellules de sortie, sinon Csound plantera avec une erreur de segmentation. En affectant zéro à cet argument, la sortie sera dirigée vers l'espace zak dans la zone de taux-k. Dans ce cas, l'espace zak doit avoir été alloué au préalable avec l'opcode *zakin* et il faut s'assurer que la taille d'allocation est suffisante pour couvrir toutes les réglettes. La valeur par défaut est zéro (c'est-à-dire stockage de la sortie dans l'espace zak).

*istart\_index* (facultatif, 0 par défaut) -- un nombre entier indiquant un décalage des positions des cellules de sortie. Il peut être positif pour permettre l'allocation en sortie de plusieurs bancs de réglettes dans la même table ou dans l'espace zak. La valeur par défaut est zéro (pas de décalage).

*iminmaxtable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste de couples min-max pour chaque réglette. Une valeur de zéro signifie l'intervalle allant de 0 à 1 pour toutes les réglettes, sans fournir de table. La valeur par défaut est zéro.

*iexptable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste d'identifiants (des nombres entiers) fournis pour modifier le comportement de chaque réglette de manière indépendante. Les identifiants peuvent avoir les valeurs suivantes :

- -1 -- courbe de réponse exponentielle
- 0 -- réponse linéaire
- nombre > 0 -- suit la courbe d'une table définie au préalable pour mettre en forme la réponse de la réglette correspondante. Dans ce cas, ce nombre correspond au numéro de la table.

On peut souhaiter que toutes les réglettes du banc aient la même courbe de réponse (exponentielle ou linéaire). Dans ce cas, on peut affecter -1 ou 0 à *iexptable* sans se préoccuper de définir auparavant une table. La valeur par défaut est zéro (toutes les réglettes ont une réponse linéaire sans avoir à fournir de

table).

*ityetable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste d'identifiants (des nombres entiers) fournis pour modifier l'aspect de chaque réglette de manière indépendante. Les identifiants peuvent avoir les valeurs suivantes :

- 0 = Réglette stylée
- 1 = Réglette pleine
- 3 = Réglette normale
- 5 = Réglette stylée
- 7 = Réglette stylée en creux

On peut souhaiter que toutes les réglettes du banc aient le même aspect. Dans ce cas, on peut affecter un nombre négatif à *ityetable* sans se préoccuper de définir auparavant une table. Les nombres négatifs ont la même signification que les identifiants positifs correspondants sauf que le même aspect est affecté à toutes les réglettes. On peut aussi donner un aspect aléatoire à chaque réglette en affectant à *ityetable* un nombre négatif inférieur à -7. La valeur par défaut est 0 (toutes les réglettes sont stylées, sans avoir à fournir de table).

On peut ajouter 20 à une valeur dans la table pour donner l'aspect "plastique" à la réglette, ou soustraire 20 si l'on veut affecter la valeur à tous les widgets sans définir une table (par exemple -21 pour donner à toutes les réglettes le type Plastique Plein).

*iwidth* (facultatif) -- largeur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées à la gauche de cette zone.

*iheight* (facultatif) -- hauteur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées à la gauche de cette zone.

*ix* (facultatif) -- position horizontale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace à la gauche de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*iy* (facultatif) -- position verticale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace à la gauche de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLslidBnk* est un widget contenant un banc de réglettes horizontales. On peut y mettre n'importe quel nombre de réglettes (argument *inumsliders*). La sortie de toutes les réglettes est stockée dans une table allouée au préalable ou dans l'espace zak (argument *ioutable*). Il est possible de déterminer la première position de la table (ou de l'espace zak) dans lequel stocker la sortie de la première réglette au moyen de l'argument *istart\_index*.

Chaque réglette peut avoir une étiquette individuelle placée à sa gauche. Les étiquettes sont définies par l'argument « *names* ». L'intervalle de sortie de chaque réglette peut être fixé individuellement au moyen d'une table externe (argument *iminmaxtable*). La courbe de réponse de chaque réglette peut être fixée individuellement, au moyen d'une liste d'identifiants placés dans une table (argument *ixptable*). Il est pos-

sible de définir l'aspect de chaque réglette indépendamment ou de donner le même aspect à toutes les réglettes (argument *ityetable*).

Les arguments *iwidth*, *iheight*, *ix* et *iy* déterminent la largeur, la hauteur, les positions horizontale et verticale de la zone rectangulaire contenant les réglettes. Noter que l'étiquette de chaque réglette est placée à sa gauche et n'est pas incluse dans la zone rectangulaire contenant les réglettes. Ainsi l'utilisateur doit laisser assez d'espace à la gauche du banc en affectant une valeur suffisante à *ix* afin que les étiquettes soient visibles.



## IMPORTANT !

Noter que les tables utilisées par *FLslidBnk* doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

## Exemples

Voici un exemple de l'opcode *FLslidBnk*. Il utilise le fichier *FLslidBnk.csd* [examples/FLslidBnk.csd].

### Exemple 177. Exemple de l'opcode *FLslidBnk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLslidBnk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

gityetable ftgen 0, 0, 8, -2, 1, 1, 3, 3, 5, 5, 7, 7
giouttable ftgen 0, 0, 8, -2, 0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1

FLpanel "Slider Bank", 400, 380, 50, 50
;Number of sliders
inum = 8
; Table to store output
iouttable = giouttable
; Width of the slider bank in pixels
iwidth = 350
; Height of the slider in pixels
iheight = 160
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 30
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 10
; Table containing fader types
ityetable = gityetable
FLslidBnk "l@2@3@4@5@6@7@8", inum , iouttable , iwidth , iheight , ix \
, iy , itypeable
FLslidBnk "l@2@3@4@5@6@7@8", inum , iouttable , iwidth , iheight , ix \
, iy + 200 , -23
; End of panel contents
FLpanelEnd
```



```
; Run the widget thread!
FLrun

instr 1
;Dummy instrument
endin

</CsInstruments>
<CsScore>

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLslider, FLslidBnk2, FLvslidBnk, FLvslidBnk2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLslidBnk2

FLslidBnk2 — Un widget FLTK contenant un banc de réglottes horizontales.

## Description

*FLslidBnk2* est un widget FLTK contenant un banc de réglottes horizontales.

## Syntaxe

**FLslidBnk2** "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart\_index]

## Initialisation

« *names* » -- une chaîne de caractères entre guillemets contenant le nom de chaque réglotte. Chaque réglotte peut avoir un nom différent. Chaque nom est séparé par un caractère « @ », par exemple : « fréquence@amplitude@coupure ». Il est possible de ne fournir aucun nom en donnant un seul espace « ». Dans ce cas, l'opcode affectera automatiquement un numéro en progression ascendante comme étiquette pour chaque réglotte.

*inumsliders* -- le nombre de réglottes.

*ioutable* (facultatif, 0 par défaut) -- numéro d'une table allouée préalablement dans laquelle seront stockées les valeurs de sortie de chaque réglotte. Il faut s'assurer que la taille de la table est suffisante pour contenir toutes les cellules de sortie, sinon Csound plantera avec une erreur de segmentation. En affectant zéro à cet argument, la sortie sera dirigée vers l'espace zak dans la zone de taux-k. Dans ce cas, l'espace zak doit avoir été alloué au préalable avec l'opcode *zakinit* et il faut s'assurer que la taille d'allocation est suffisante pour couvrir toutes les réglottes. La valeur par défaut est zéro (c'est-à-dire stockage de la sortie dans l'espace zak).

*iconfigtable* -- dans les opcodes *FLslidBnk2* et *FLvslidBnk2*, cette table remplace *iminmaxtable*, *iexp-table* et *istyletable*, tous ces paramètres étant placés dans une seule table. Cette table doit être remplie avec un groupe de quatre paramètres pour chaque réglotte de la façon suivante :

min1, max1, exp1, style1, min2, max2, exp2, style2, min3, max3, exp3, style3 etc.

par exemple en utilisant GEN02 on peut taper :

*inumftgen* 1,0,256, -2, 0,1,0,1, 100, 5000, -1, 3, 50, 200, -1, 5,..... [etc]

Dans cet exemple la première réglotte reçoit les paramètres [0, 1, 0, 1] (valeurs comprises entre 0 et 1, réponse linéaire, aspect réglotte pleine), la seconde réglotte reçoit les paramètres [100, 5000, -1, 3] (valeurs comprises entre 100 et 5000, réponse exponentielle, aspect réglotte normale), la troisième réglotte reçoit les paramètres [50, 200, -1, 5] (valeurs comprises entre 50 et 200, réponse exponentielle, aspect réglotte stylée), et ainsi de suite.

*iwidth* (facultatif) -- largeur de la zone rectangulaire contenant toutes les réglottes du banc, à l'exclusion des étiquettes qui sont placées à la gauche de cette zone.

*iheight* (facultatif) -- hauteur de la zone rectangulaire contenant toutes les réglottes du banc, à l'exclusion des étiquettes qui sont placées à la gauche de cette zone.

*ix* (facultatif) -- position horizontale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace à la gauche de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*iy* (facultatif) -- position verticale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace à la gauche de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*istart\_index* (facultatif, 0 par défaut) -- un nombre entier indiquant un décalage des positions des cellules de sortie. Il peut être positif pour permettre l'allocation en sortie de plusieurs bancs de réglettes dans la même table ou dans l'espace zak. La valeur par défaut est zéro (pas de décalage).

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLslidBnk2* est un widget contenant un banc de réglettes horizontales. On peut y mettre n'importe quel nombre de réglettes (argument *inumsliders*). La sortie de toutes les réglettes est stockée dans une table allouée au préalable ou dans l'espace zak (argument *ioutable*). Il est possible de déterminer la première position de la table (ou de l'espace zak) dans laquelle stocker la sortie de la première réglette au moyen de l'argument *istart\_index*.

Chaque réglette peut avoir une étiquette individuelle placée à sa gauche. Les étiquettes sont définies par l'argument « *names* ». L'intervalle de sortie de chaque réglette peut être fixé individuellement au moyen des valeurs *min* et *max* dans la table *iconfigtable*. La courbe de réponse de chaque réglette peut être fixée individuellement, au moyen d'une liste d'identifiants placés dans la table *iconfigtable* (argument *exp*). Il est possible de définir l'aspect de chaque réglette indépendamment ou de donner le même aspect à toutes les réglettes (argument *style* dans la table *iconfigtable*).

Les arguments *iwidth*, *iheight*, *ix* et *iy* déterminent la largeur, la hauteur, les positions horizontale et verticale de la zone rectangulaire contenant les réglettes. Noter que l'étiquette de chaque réglette est placée à sa gauche et n'est pas incluse dans la zone rectangulaire contenant les réglettes. Ainsi l'utilisateur doit laisser assez d'espace à la gauche du banc en affectant une valeur suffisante à *ix* afin que les étiquettes soient visibles.



### IMPORTANT !

Noter que les tables utilisées par *FLslidBnk2* doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

## Exemples

Voici un exemple de l'opode *FLslidBnk2*. Il utilise le fichier *FLslidBnk2.csd* [examples/FLslidBnk2.csd].

### Exemple 178. Exemple de l'opode *FLslidBnk2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          -M0 ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 100
nchnls = 2

;Example by Gabriel Maldonado

giElem init 8
giOutTab ftgen 1,0,128, 2,          0

;min1, max1, exp1, type1, min2, max2, exp2, type2, min3, max3, exp3, type3 etc.
giConfigTab ftgen 2,0,128,-2,          .1, 1000, -1, 3,          .1, 1000, -1, 3,          .1, 1000, -1, 3,          3
;min1, max1, exp1, type1, min2, max2, exp2, type2, min3, max3, exp3, type3 etc.
;          .1, 5000, -1, 5,          .1, 5000, -1, 5,          .1, 5000, -1, 5,          5
giSine ftgen 3,0,256,10, 1

FLpanel "This Panel contains a Slider Bank",600,600
FLslidBnk2 "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab, giConfigTab, 400, 5
FLpanel_end

FLrun

instr 1

kmodindex1          init 0
kmodindex2          init 0
kmodindex3          init 0
kamp                init 0
kfreq1              init 0
kfreq2              init 0
kfreq3              init 0
kfreq4              init 0

vtablelk giOutTab, kmodindex1 , kmodindex2, kmodindex3, kamp, kfreq1, kfreq2 , kfreq3, kfreq4

amod1 oscili kmodindex1, kfreq1, giSine
amod2 oscili kmodindex2, kfreq2, giSine
amod3 oscili kmodindex3, kfreq3, giSine
aout oscili kamp,          kfreq4+amod1+amod2+amod3, giSine

outs aout, aout
endin

</CsInstruments>
<CsScore>

i1 0 3600
f0 3600

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLslider, FLslidBnk, FLvslidBnk, FLvslidBnk2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslidBnkGetHandle

FLslidBnkGetHandle — récupère l'identifiant du dernier banc de réglettes créé.

## Description

*FLslidBnkGetHandle* récupère l'identifiant du dernier banc de réglettes créé.

## Syntaxe

`ihandle` **FLslidBnkGetHandle**

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLslidBnkGetHandle* récupère l'identifiant du dernier banc de réglettes créé. Cet opcode doit suivre immédiatement un opcode *FLslidBnk* (ou *FLvslidBnk*, *FLslidBnk2* et *FLvslidBnk2*), afin de récupérer son identifiant.

Voir l'entrée *FLslidBnk2Setk* pour un exemple d'utilisation.

## Voir Aussi

*FLslider*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnk2Set*, *FLslidBnk2Setk*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslidBnkSet

FLslidBnkSet — modifie les valeurs d'un banc de réglettes.

## Description

*FLslidBnkSet* modifie les valeurs d'un banc de réglettes selon un ensemble de valeurs stockées dans une table.

## Syntaxe

```
FLslidBnkSet ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

*ifn* - numéro d'une table contenant un ensemble de valeurs à affecter à chaque réglette.

*istartIndex* - (facultatif) indice dans la table du premier élément à être évalué. La valeur par défaut est zéro.

*istartSlid* - (facultatif) première réglette à évaluer. 0 par défaut, indiquant la première réglette.

*inumSlid* - (facultatif) nombre de réglettes à mettre à jour. 0 par défaut, indiquant toutes les réglettes.

## Exécution

*FLslidBnkSet* modifie les valeurs d'un banc de réglettes (créé avec *FLslidBnk* ou avec *FLvslidBnk*) selon un ensemble de valeurs stockées dans la table *ifn*. Il permet actuellement de mettre à jour un banc de réglettes *FLslidBnk* (ou *FLvslidBnk*), (par exemple en utilisant l'opcode *slider8table*) avec un ensemble de valeurs situées dans une table. Il faut mettre dans l'argument *ihandle* l'identifiant reçu de l'opcode *FLslidBnkGetHandle*. Il ne travaille qu'au taux-i. Il est possible de ne réinitialiser qu'une partie des réglettes, en utilisant les arguments facultatifs *istartIndex*, *istartSlid*, *inumSlid*.

Il y a une version de taux-k de cet opcode appelée *FLslidBnkSetk*.

## Voir Aussi

*FLslider*, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnk2Set*, *FLslidBnkSetk*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslidBnkSetk

FLslidBnkSetk — modifie les valeurs d'un banc de réglettes.

## Description

*FLslidBnkSetk* modifie les valeurs d'un banc de réglettes selon un ensemble de valeurs stockées dans une table.

## Syntaxe

```
FLslidBnkSetk ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

*ifn* - numéro d'une table contenant un ensemble de valeurs à affecter à chaque réglette.

*istartIndex* - (facultatif) indice dans la table du premier élément à être évalué. La valeur par défaut est zéro.

*istartSlid* - (facultatif) première réglette à évaluer. 0 par défaut, indiquant la première réglette.

*inumSlid* - (facultatif) nombre de réglettes à mettre à jour. 0 par défaut, indiquant toutes les réglettes.

## Exécution

*ktrig* – la sortie de *FLslidBnkSetk* est un déclencheur qui indique si les réglettes doivent être mises à jour ou pas. Une valeur non nulle force la mise à jour des réglettes.

*FLslidBnkSetk* est semblable à *FLslidBnkSet* mais il permet de modifier les valeurs de *FLslidBnk* au taux-k (on peut aussi utiliser *FLslidBnkSetk* avec *FLvslidBnk*, obtenant un résultat identique). Il permet aussi de relier le banc de réglettes au MIDI. Si l'on utilise le MIDI (par exemple au moyen de l'opcode *slider8table*), *FLslidBnkSetk* change les valeurs du banc de réglettes *FLslidBnk* avec un ensemble de valeurs situées dans une table. Cet opcode est ainsi capable de servir de pont MIDI vers le widget *FLslidBnk* lorsqu'il est utilisé avec la famille d'opcodes *sliderXXtable* (voir l'entrée *slider8table* pour plus d'information). Noter que, si l'on veut utiliser l'indexation de table comme une courbe de réponse, il est impossible de le faire directement dans la configuration *iconfigtable* de *FLslidBnk2*, lorsque l'on a l'intention d'utiliser l'opcode *FLslidBnkSetk*. En fait, l'élément correspondant de l'élément *inputTable* de *FLslidBnkSetk* doit être positionné en mode linéaire et respecter l'intervalle de 0 à 1. Même les éléments correspondants de *sliderXXtable* doivent être positionnés en mode linéaire dans l'intervalle normalisé. On peut indexer la table plus tard au moyen des opcodes *tab* et *tb*, et recadrer la sortie en fonction des valeurs max et min. D'un autre côté, il est possible d'utiliser une courbe de réponse linéaire ou exponentielle directement, en fixant l'intervalle min-max courant ainsi que l'indicateur à la fois dans l'*iconfigtable* du *FLslidBnk2* correspondant et dans *sliderXXtable*.

*FLslidBnkSetk* est la version de taux-k de *FLslidBnk2Set*.

## Voir Aussi

*FLslider*, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnkSet*, *FLs-*

*lidBnk2Set, slider8table*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06



# FLslidBnk2Set

FLslidBnk2Set — modifie les valeurs d'un banc de réglettes.

## Description

*FLslidBnk2Set* modifie les valeurs d'un banc de réglettes selon un ensemble de valeurs stockées dans une table.

## Syntaxe

```
FLslidBnk2Set ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

*ifn* - numéro d'une table contenant un ensemble de valeurs à affecter à chaque réglette.

*istartIndex* - (facultatif) indice dans la table du premier élément à être évalué. La valeur par défaut est zéro.

*istartSlid* - (facultatif) première réglette à évaluer. 0 par défaut, indiquant la première réglette.

*inumSlid* - (facultatif) nombre de réglettes à mettre à jour. 0 par défaut, indiquant toutes les réglettes.

## Exécution

*FLslidBnk2Set* modifie les valeurs d'un banc de réglettes (créé avec *FLslidBnk2* ou avec *FLvslidBnk2*) selon un ensemble de valeurs stockées dans la table *ifn*. Il permet actuellement de mettre à jour un banc de réglettes *FLslidBnk2* (ou *FLvslidBnk2*), (par exemple en utilisant l'opcode *slider8table*) avec un ensemble de valeurs situées dans une table. Il faut mettre dans l'argument *ihandle* l'identifiant reçu de l'opcode *FLslidBnkGetHandle*. Il ne travaille qu'au taux-i. Il est possible de ne réinitialiser qu'une partie des réglettes, en utilisant les arguments facultatifs *istartIndex*, *istartSlid*, *inumSlid*.

*FLslidBnk2Set* est identique à *FLslidBnkSet*, mais il travaille sur *FLslidBnk2* et *FLvslidBnk2* au lieu de *FLslidBnk* et de *FLvslidBnk*.

Il y a une version de taux-k de cet opcode appelée *FLslidBnk2Setk*.

## Voir Aussi

*FLslider*, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnkSet*, *FLslidBnk2Setk*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslidBnk2Setk

FLslidBnk2Setk — modifie les valeurs d'un banc de réglettes.

## Description

*FLslidBnk2Setk* modifie les valeurs d'un banc de réglettes selon un ensemble de valeurs stockées dans une table.

## Syntaxe

```
FLslidBnk2Setk ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

*ifn* - numéro d'une table contenant un ensemble de valeurs à affecter à chaque réglette.

*istartIndex* - (facultatif) indice dans la table du premier élément à être évalué. La valeur par défaut est zéro.

*istartSlid* - (facultatif) première réglette à évaluer. 0 par défaut, indiquant la première réglette.

*inumSlid* - (facultatif) nombre de réglettes à mettre à jour. 0 par défaut, indiquant toutes les réglettes.

## Exécution

*ktrig* – la sortie de *FLslidBnk2Setk* est un déclencheur qui indique si les réglettes doivent être mises à jour ou pas. Une valeur non nulle force la mise à jour des réglettes.

*FLslidBnk2Setk* est semblable à *FLslidBnkSet* mais il permet de modifier les valeurs de *FLslidBnk2* au taux-k (on peut aussi utiliser *FLslidBnk2Setk* avec *FLvslidBnk2*, obtenant un résultat identique). Il permet aussi de relier le banc de réglettes au MIDI. Si l'on utilise le MIDI (par exemple au moyen de l'opcode *slider8table*), *FLslidBnk2Setk* change les valeurs du banc de réglettes *FLslidBnk2* avec un ensemble de valeurs situées dans une table. Cet opcode est ainsi capable de servir de pont MIDI vers le widget *FLslidBnk2* lorsqu'il est utilisé avec la famille d'opcodes *sliderXXtable* (voir l'entrée *slider8table* pour plus d'information). Noter que, si l'on veut utiliser l'indexation de table comme une courbe de réponse, il est impossible de le faire directement dans la configuration *iconfigtable* de *FLslidBnk2*, lorsque l'on a l'intention d'utiliser l'opcode *FLslidBnk2Setk*. En fait, l'élément correspondant de l'élément *input-Table* de *FLslidBnk2Setk* doit être positionné en mode linéaire et respecter l'intervalle de 0 à 1. Même les éléments correspondants de *sliderXXtable* doivent être positionnés en mode linéaire dans l'intervalle normalisé. On peut indexer la table plus tard au moyen des opcodes *tab* et *tb*, et recadrer la sortie en fonction des valeurs max et min. D'un autre côté, il est possible d'utiliser une courbe de réponse linéaire ou exponentielle directement, en fixant l'intervalle min-max courant ainsi que l'indicateur à la fois dans l'*iconfigtable* du *FLslidBnk2* correspondant et dans *sliderXXtable*.

*FLslidBnk2Setk* est la version de taux-k de *FLslidBnk2Set*.

## Exemples

Voici un exemple de l'opcode *FLslidBnk2Setk*. Il utilise le fichier *FLslidBnk2Setk.csd* [exemples/FLs-

lidBnk2Setk.csd].

## Exemple 179. Exemple de l'opcode FLslidBnk2Setk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out   Audio in   No messages
-odac         -iadc      -d          ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr          = 44100
ksmps      = 10
nchnls     = 2

;Example by Gabriel Maldonado 2007
giElem init 8
giOutTab ftgen 1,0,128, 2,          0
giSine ftgen 3,0,256,10, 1
giOutTab2 ftgen 4,0,128, 2,          0
itab ftgen 29, 0, 129, 5, .002, 128, 1          ;** exponential ascending curve for slider mapping
giExpTab ftgen 30, 0, 129, -24, itab, 0, 1          ;** rescaled curve for slider mapping
giConfigTab ftgen 2,0,128,-2,          1,          500, -1,          13, \
          1,          500, -1,          13, \
          1,          500, -1,          13, \
          1,          5000, -1,          13, \
\
          1,          1000,          -1,          5, \
          1,          1000,          -1,          5, \
          1,          1000,          -1,          5, \
          1,          5000,          -1,          5

FLpanel "Multiple FM",600,600
FLslidBnk2 "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab2, giConfigTab, 400,
giHandle FLslidBnkGetHandle
FLpanel_end

FLrun
instr 1
ktrig slider8table 1, giOutTab, 0, \ ; ctl min max init func
27, 1,          500, 3, -1, \ ;1 repeat rate
28, 1,          500, 4, -1, \ ;2 random freq. amount
29, 1,          500, 1, -1, \ ;3 random amp. amount
30, 1, 5000, 1, -1, \ ;4 number of concurrent loop points
\
31, 1,          1000, 1, -1, \ ;5 kloop1
32, 1,          1000, 1, -1, \ ;6 kloop2
33, 1,          1000, 1, -1, \ ;7 kloop3
34, 1,          1000, 1, -1, \ ;8 kloop4
kmodindex1 init 0
kmodindex2 init 0
kmodindex3 init 0
kamp init 0
kfreq1 init 0
kfreq2 init 0
kfreq3 init 0
kfreq4 init 0
vtablelk giOutTab2, kmodindex1, kmodindex2, kmodindex3, kamp, kfreq1, kfreq2, kfreq3, kfreq4
; *kflag, *ihandle, *ifn, *startInd, *startSlid, *numSlid;
FLslidBnk2Setk ktrig, giHandle, giOutTab, 0, 0, giElem
printk2 kmodindex1
printk2 kmodindex2,10
printk2 kmodindex3,20
printk2 kamp,30
amod1 oscili kmodindex1, kfreq1, giSine
amod2 oscili kmodindex2, kfreq2, giSine
amod3 oscili kmodindex3, kfreq3, giSine
aout oscili kamp,          kfreq4+amod1+amod2+amod3, giSine
outs aout, aout
endin
</CsInstruments>
<CsScore>
i1 0 3600
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*FLslider, FLslidBnkGetHandle, FLslidBnk, FLslidBnk2, FLvslidBnk, FLvslidBnk2 FLslidBnkSet, FLslidBnk2Set, slider8table*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslider

FLslider — Dispose une réglette dans le conteneur FLTK correspondant.

## Description

*FLslider* dispose une réglette dans le conteneur correspondant.

## Syntaxe

```
kout, ihandle FLslider "label", imin, imax, iexp, itype, idisp, iwidth, \
    iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLslider* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- valeur minimale de l'intervalle de sortie (correspond à la valeur à gauche pour les réglettes horizontales et à la valeur du haut pour les réglettes verticales).

*imax* -- valeur maximale de l'intervalle de sortie (correspond à la valeur à droite pour les réglettes horizontales et à la valeur du bas pour les réglettes verticales).

L'argument *imin* peut être supérieur à l'argument *imax*. Cela a pour effet d'« inverser » l'objet si bien que les valeurs supérieures sont dans la direction opposée. L'extrémité remplie des réglettes pleines est aussi inversée.

*iexp* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexp* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.



### IMPORTANT !

Noter que les tables utilisées par les valuateurs doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

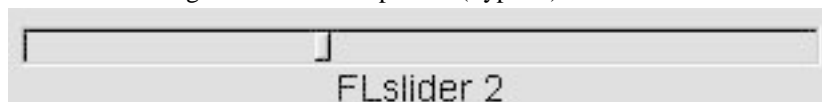
*itype* -- un nombre entier indiquant l'apparence du valuateur.

L'argument *itype* accepte les valeurs suivantes :

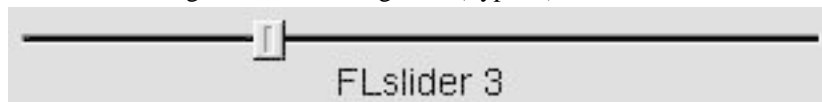
- 1 - une réglette horizontale pleine
- 2 - une réglette verticale pleine
- 3 - une réglette horizontale gravée
- 4 - une réglette verticale gravée
- 5 - une réglette horizontale stylée
- 6 - une réglette verticale stylée
- 7 - une réglette horizontale stylée saillante
- 8 - une réglette verticale stylée saillante



FLslider - une réglette horizontale pleine (itype=1).



FLslider - une réglette horizontale gravée (itype=3).



FLslider - une réglette horizontale stylée (itype=5).

On peut aussi créer des réglettes à l'aspect "plastique" en ajoutant 20 à *itype*.

*idisp* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*kout* -- valeur en sortie.

Les réglettes sont créées avec la valeur minimale située par défaut à gauche/en haut. Si l'on veut inverser la réglette, il faut inverser les valeurs. Voir l'exemple ci-dessous.

## Exemples

Voici un exemple de l'opcode FLslider. Il utilise le fichier *FLslider.csd* [examples/FLslider.csd].

### Exemple 180. Exemple de l'opcode FLslider.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLslider.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flslider controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Slider", 900, 400, 50, 50
; Minimum value output by the slider
imin = 200
; Maximum value output by the slider
imax = 5000
; Logarithmic type slider selected
iexp = -1
; Slider graphic type (5='nice' slider)
itype = 5
; Display handle (-1=not used)
idisp = -1
; Width of the slider in pixels
iwidth = 750
; Height of the slider in pixels
iheight = 30
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 125
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 50

gkfreq, ihandle FLslider "Frequency", imin, imax, iexp, itype, idisp, iwidth, iheight, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

;Set the widget's initial value
FLsetVal_i 300, ihandle

instr 1
iamp = 15000
ifn = 1
kfreq portk gkfreq, 0.005 ;Smooth gkfreq to avoid zipper noise
asig oscili iamp, kfreq, ifn
out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
```

e

```
</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode FLslider, montrant les types de réglettes et d'autres options. Il montre aussi l'utilisation de FLvalue pour afficher le contenu d'un widget. Il utilise le fichier FLslider-2.csd [exemples/FLslider-2.csd].

### Exemple 181. Exemple plus complexe de l'opcode FLslider.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadac      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLslider-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

;By Andres Cabrera 2007

FLpanel "Slider Types", 410, 260, 50, 50
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 10
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 10
; Create boxes to display widget values
gvalue1 FLvalue "1", 60, 20, ix + 330, iy
gvalue3 FLvalue "3", 60, 20, ix + 330, iy + 40
gvalue5 FLvalue "5", 60, 20, ix + 330, iy + 80

gvalue2 FLvalue "2", 60, 20, ix + 60, iy + 140
gvalue4 FLvalue "4", 60, 20, ix + 195, iy + 140
gvalue6 FLvalue "6", 60, 20, ix + 320, iy + 140

;Horizontal sliders
gkdummy1, gihandle1 FLslider "Type 1", 200, 5000, -1, 1, gvalue1, 320, 20, ix, iy
gkdummy3, gihandle3 FLslider "Type 3", 0, 15000, 0, 3, gvalue3, 320, 20, ix, iy + 40
; Reversed slider
gkdummy5, gihandle5 FLslider "Type 5", 1, 0, 0, 5, gvalue5, 320, 20, ix, iy + 80

;Vertical sliders
gkdummy2, gihandle2 FLslider "Type 2", 0, 1, 0, 2, gvalue2, 20, 100, ix+ 30 , iy + 120
; Reversed slider
gkdummy4, gihandle4 FLslider "Type 4", 1, 0, 0, 4, gvalue4, 20, 100, ix + 165 , iy + 120
gkdummy6, gihandle6 FLslider "Type 6", 0, 1, 0, 6, gvalue6, 20, 100, ix + 290 , iy + 120
FLpanelEnd

FLpanel "Plastic Slider Types", 410, 300, 150, 150
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 10
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 10
; Create boxes to display widget values
gvalue21 FLvalue "21", 60, 20, ix + 330, iy
gvalue23 FLvalue "23", 60, 20, ix + 330, iy + 40
gvalue25 FLvalue "25", 60, 20, ix + 330, iy + 80

gvalue22 FLvalue "22", 60, 20, ix + 60, iy + 140
gvalue24 FLvalue "24", 60, 20, ix + 195, iy + 140
gvalue26 FLvalue "26", 60, 20, ix + 320, iy + 140
```



```

;Horizontal sliders
gkdummy21, gihandle21 FLslider "Type 21", 200, 5000, -1, 21, givalue21, 320, 20, ix, iy
gkdummy23, gihandle23 FLslider "Type 23", 0, 15000, 0, 23, givalue23, 320, 20, ix, iy + 40
; Reversed slider
gkdummy25, gihandle25 FLslider "Type 25", 1, 0, 0, 25, givalue25, 320, 20, ix, iy + 80

;Vertical sliders
gkdummy22, gihandle22 FLslider "Type 22", 0, 1, 0, 22, givalue22, 20, 100, ix+ 30 , iy + 120
; Reversed slider
gkdummy24, gihandle24 FLslider "Type 24", 1, 0, 0, 24, givalue24, 20, 100, ix + 165 , iy + 120
gkdummy26, gihandle26 FLslider "Type 26", 0, 1, 0, 26, givalue26, 20, 100, ix + 290 , iy + 120
;Button to add color to the sliders
gkcolors, ihdummy FLbutton "Color", 1, 0, 21, 150, 30, 30, 260, 0, 10, 0, 1
FLpanelEnd
FLrun

;Set some widget's initial value
FLsetVal_i 500, gihandle1
FLsetVal_i 1000, gihandle3

instr 10
; Set the color of widgets
FLsetColor 200, 230, 0, gihandle1
FLsetColor 0, 123, 100, gihandle2
FLsetColor 180, 23, 12, gihandle3
FLsetColor 10, 230, 0, gihandle4
FLsetColor 0, 0, 0, gihandle5
FLsetColor 0, 0, 0, gihandle6

FLsetColor 200, 230, 0, givalue1
FLsetColor 0, 123, 100, givalue2
FLsetColor 180, 23, 12, givalue3
FLsetColor 10, 230, 0, givalue4
FLsetColor 255, 255, 255, givalue5
FLsetColor 255, 255, 255, givalue6

FLsetColor2 20, 23, 100, gihandle1
FLsetColor2 200,0 ,123 , gihandle2
FLsetColor2 180, 180, 100, gihandle3
FLsetColor2 180, 23, 12, gihandle4
FLsetColor2 180, 180, 100, gihandle5
FLsetColor2 180, 23, 12, gihandle6

FLsetColor 200, 230, 0, gihandle21
FLsetColor 0, 123, 100, gihandle22
FLsetColor 180, 23, 12, gihandle23
FLsetColor 10, 230, 0, gihandle24
FLsetColor 0, 0, 0, gihandle25
FLsetColor 0, 0, 0, gihandle26

FLsetColor 200, 230, 0, givalue21
FLsetColor 0, 123, 100, givalue22
FLsetColor 180, 23, 12, givalue23
FLsetColor 10, 230, 0, givalue24
FLsetColor 255, 255, 255, givalue25
FLsetColor 255, 255, 255, givalue26

FLsetColor2 20, 23, 100, gihandle21
FLsetColor2 200,0 ,123 , gihandle22
FLsetColor2 180, 180, 100, gihandle23
FLsetColor2 180, 23, 12, gihandle24
FLsetColor2 180, 180, 100, gihandle25
FLsetColor2 180, 23, 12, gihandle26

; Slider values must be updated for colors to change
FLsetVal_i 250, gihandle1
FLsetVal_i 0.5, gihandle2
FLsetVal_i 0, gihandle3
FLsetVal_i 0, gihandle4
FLsetVal_i 0, gihandle5
FLsetVal_i 0.5, gihandle6
FLsetVal_i 250, gihandle21
FLsetVal_i 0.5, gihandle22
FLsetVal_i 500, gihandle23
FLsetVal_i 0, gihandle24
FLsetVal_i 0, gihandle25
FLsetVal_i 0.5, gihandle26

```

```
endin

</CsInstruments>
<CsScore>
f 0 3600 ;Dumy table to make csound wait for realtime events
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLcount, FLjoy, FLknob, FLroller, FLslidBnk, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Février 2004. Grâce à une note de Dave Phillips, le paramètre étranger *istep* a été effacé.

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLtabs

FLtabs — Crée une interface FLTK à onglets.

## Description

*FLtabs* est une interface à onglets qui est utile pour afficher alternativement plusieurs zones contenant des widgets dans la même fenêtre. Il doit être utilisé en même temps qu'un *FLgroup*, un autre conteneur qui regroupe des widgets enfants.

## Syntaxe

```
FLtabs iwidth, iheight, ix, iy
```

## Initialisation

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphique des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

*FLtabs* est une interface à onglets qui est utile pour afficher alternativement plusieurs zones contenant des widgets dans la même fenêtre.



FLtabs.

Il doit être utilisé en même temps qu'un *FLgroup*, un autre opcode de conteneur FLTK qui regroupe des widgets enfants.

## Exemples

Le code de l'exemple suivant :

```

FLpanel "Panel1", 450, 550, 100, 100
FLscroll 450, 550, 0, 0
FLtabs 400, 550, 5, 5

FLgroup "sliders", 380, 500, 10, 40, 1
gk1, ihs FLslider "FLslider 1", 500, 1000, 2, 1, -1, 300, 15, 20, 50
gk2, ihs FLslider "FLslider 2", 300, 5000, 2, 3, -1, 300, 15, 20, 100
gk3, ihs FLslider "FLslider 3", 350, 1000, 2, 5, -1, 300, 15, 20, 150
gk4, ihs FLslider "FLslider 4", 250, 5000, 1, 11, -1, 300, 30, 20, 200
gk5, ihs FLslider "FLslider 5", 220, 8000, 2, 1, -1, 300, 15, 20, 250
gk6, ihs FLslider "FLslider 6", 1, 5000, 1, 13, -1, 300, 15, 20, 300
gk7, ihs FLslider "FLslider 7", 870, 5000, 1, 15, -1, 300, 30, 20, 350
gk8, ihs FLslider "FLslider 8", 20, 20000, 2, 6, -1, 30, 400, 350, 50
FLgroupEnd

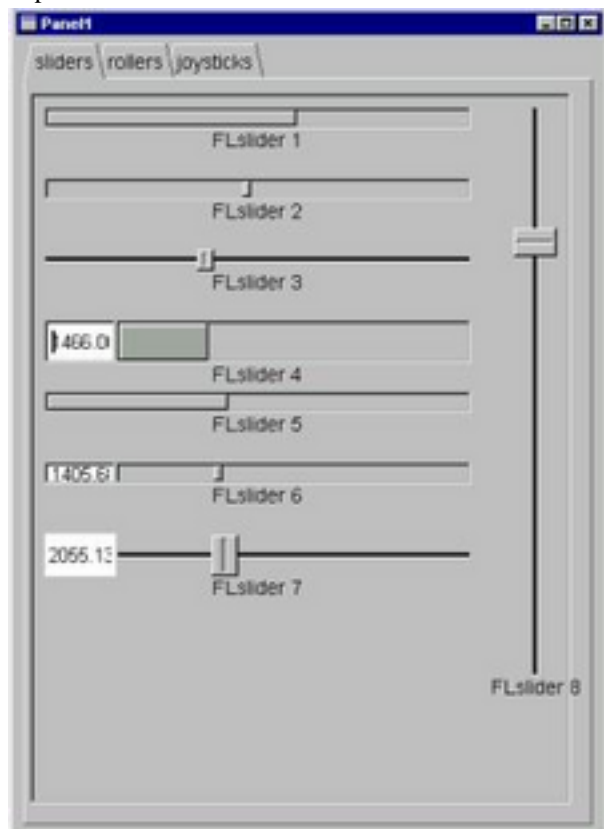
FLgroup "rollers", 380, 500, 10, 30, 2
gk1, ihr FLroller "FLroller 1", 50, 1000, 1, 2, 1, -1, 200, 22, 20, 50
gk2, ihr FLroller "FLroller 2", 80, 5000, 1, 2, 1, -1, 200, 22, 20, 100
gk3, ihr FLroller "FLroller 3", 50, 1000, 1, 2, 1, -1, 200, 22, 20, 150
gk4, ihr FLroller "FLroller 4", 80, 5000, 1, 2, 1, -1, 200, 22, 20, 200
gk5, ihr FLroller "FLroller 5", 50, 1000, 1, 2, 1, -1, 200, 22, 20, 250
gk6, ihr FLroller "FLroller 6", 80, 5000, 1, 2, 1, -1, 200, 22, 20, 300
gk7, ihr FLroller "FLroller 7", 50, 5000, 1, 1, 2, -1, 30, 300, 280, 50
FLgroupEnd

FLgroup "joysticks", 380, 500, 10, 40, 3
gk1, gk2, ihj1, ihj2 FLjoy "FLjoy", 50, 18000, 50, 18000, 2, 2, -1, -1, 300, 300, 30, 60
FLgroupEnd

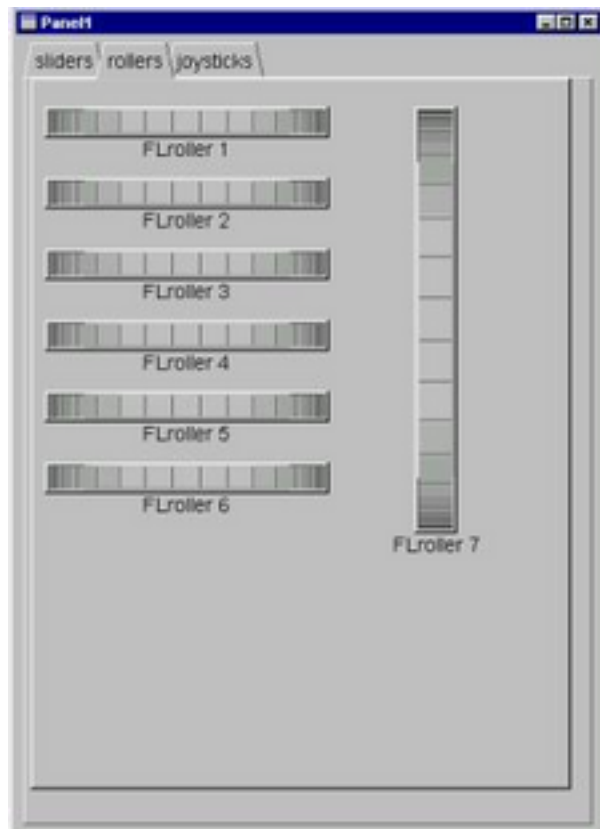
FLtabsEnd
FLscrollEnd
FLpanelEnd

```

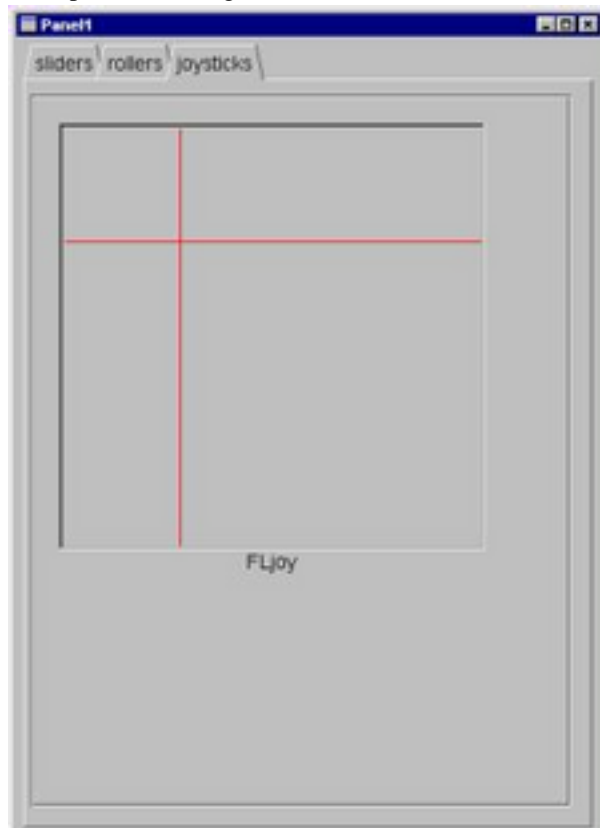
... produira le résultat suivant :



Exemple FLtabs, onglet des réglottes.



Exemple FLtabs, onglet des molettes.



Exemple FLtabs, onglet du joystick.

(Chaque image montre un onglet différent sélectionné dans la même fenêtre.)

## Exemples

Voici un exemple de l'opcode FLtabs. Il utilise le fichier *FLtabs.csd* [exemples/FLtabs.csd].

### Exemple 182. Exemple de l'opcode FLtabs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLtabs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A single oscillator with frequency, amplitude and
; panning controls on separate file tab cards
sr = 44100
kr = 441
ksmps = 100
nchnls = 2

FLpanel "Tabs", 300, 350, 100, 100
itabswidth = 280
itabsheight = 330
ix = 5
iy = 5
FLtabs itabswidth,itabsheight, ix,iy

    itablwidth = 280
    itablheight = 300
    itablx = 10
    itably = 40
    FLgroup "Tab 1", itablwidth, itablheight, itablx, itably
        gkfreq, i1 FLknob "Frequency", 200, 5000, -1, 1, -1, 70, 70, 130
        FLsetVal_i 400, i1
    FLgroupEnd

    itab2width = 280
    itab2height = 300
    itab2x = 10
    itab2y = 40
    FLgroup "Tab 2", itab2width, itab2height, itab2x, itab2y
        gkamp, i2 FLknob "Amplitude", 0, 15000, 0, 1, -1, 70, 70, 130
        FLsetVal_i 15000, i2
    FLgroupEnd

    itab3width = 280
    itab3height = 300
    itab3x = 10
    itab3y = 40
    FLgroup "Tab 3", itab3width, itab3height, itab3x, itab3y
        gkpan, i3 FLknob "Pan position", 0, 1, 0, 1, -1, 70, 70, 130
        FLsetVal_i 0.5, i3
    FLgroupEnd
FLtabsEnd
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    ifn = 1
    asig oscili gkamp, gkfreq, ifn
    outs asig*(1-gkpan), asig*gkpan
endin
```

```
</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLgroup, FLgroupEnd, FLpack, FLpackEnd, FLpanel, FLpanelEnd, FLscroll, FLscrollEnd, FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLtabsEnd

FLtabsEnd — Marque la fin d'une interface FLTK à onglets.

## Description

Marque la fin d'une interface FLTK à onglets.

## Syntaxe

`FLtabsEnd`

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir Aussi

*FLgroup*, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLtabs\_end

FLtabs\_end — Marque la fin d'une interface FLTK à onglets.

## Description

Marque la fin d'une interface FLTK à onglets. C'est un autre nom pour **FLtabsEnd** fourni pour des raisons de compatibilité. Voir *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLtext

FLtext — Un opcode de widget FLTK qui crée une boîte de texte.

## Description

*FLtext* permet à l'utilisateur de modifier la valeur d'un paramètre en la tapant directement dans un champ de texte.

## Syntaxe

```
kout, ihandle FLtext "label", imin, imax, istep, itype, iwidth, \  
iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLtext* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- valeur minimale de l'intervalle de sortie.

*imax* -- valeur maximale de l'intervalle de sortie.

*istep* -- un nombre en virgule flottante indiquant le pas d'incrémement du valuateur correspondant au glissé de souris. L'argument *istep* permet de ralentir le glissé autorisant une précision arbitraire.

*itype* -- un nombre entier indiquant l'apparence du valuateur.

L'argument *itype* accepte les valeurs suivantes :

- 1 - comportement normal
- 2 - l'opération du glissé de souris est supprimée, deux boutons fléchés la remplacent. Un clic de souris sur un de ces boutons peut accroître/diminuer la valeur en sortie.
- 3 - l'édition du texte est supprimée, seul le glissé de souris modifie la valeur en sortie.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

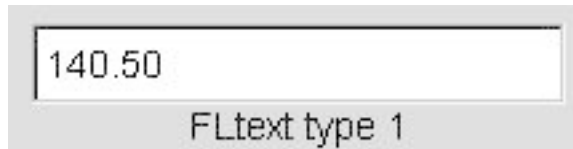
*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*kout* -- valeur en sortie.

*FLtext* permet à l'utilisateur de modifier la valeur d'un paramètre en la tapant directement dans un champ de texte.



*FLtext*.

On peut aussi modifier sa valeur en le cliquant et en glissant la souris horizontalement. L'argument *istep* permet à l'utilisateur de fixer arbitrairement la réponse au glissé de souris.

## Exemples

Voici un exemple de l'opcode *FLtext*. Il utilise le fichier *FLtext.csd* [examples/FLtext.csd].

### Exemple 183. Exemple de l'opcode *FLtext*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac -iadc -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLtext.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with fltext box controlled
; frequency either click and drag or double click and
; type to change frequency value
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Text Box", 270, 600, 50, 50
; Minimum value output by the text box
imin = 200
; Maximum value output by the text box
imax = 5000
; Step size
istep = 1
; Text box graphic type
itype = 1
; Width of the text box in pixels
iwidth = 70
; Height of the text box in pixels
iheight = 30
; Distance of the left edge of the text box
; from the left edge of the panel
ix = 100
; Distance of the top edge of the text box
; from the top edge of the panel
iy = 300

gkfreq,ihandle FLtext "Enter the frequency", imin, imax, istep, itype, iwidth, iheight, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
```

```
FLrun

instr 1
  iamp = 15000
  ifn = 1
  asig oscili iamp, gkfreq, ifn
  out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLcount, FLjoy, FLknob, FLroller, FLslider*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLupdate

FLupdate — Identique à l'opcode FLrun.

## Description

Identique à l'opcode *FLrun*.

## Syntaxe

**FLupdate**

# fluidAllOut

fluidAllOut — Rassemble toutes les données audio depuis tous les moteurs Fluidsynth dans une exécution.

## Syntaxe

`aleft, aright fluidAllOut`

## Description

Rassemble toutes les données audio depuis tous les moteurs Fluidsynth dans une exécution.

## Exécution

*aleft* -- Canal de sortie audio gauche.

*aright* -- Canal de sortie audio droite.

Appelez fluidAllOut dans une définition d'instrument dont le numéro est supérieur à ceux de toutes les définitions d'instrument de contrôle de fluid. Tous les SoundFonts envoient leur sortie audio à cet opcode. Envoyez une note de durée indéterminée à cet instrument afin d'activer les SoundFonts pour une durée suffisante.

Dans cette implémentation, les effets SoundFont tels que chorus ou réverbération sont utilisés si et seulement s'ils sont présents par défaut pour le preset. Il n'y a aucun moyen d'activer ou d'arrêter de tels effets, ou de changer leurs paramètres, depuis Csound.

## Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluidAllOut.orc* [examples/fluidAllOut.orc].

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
odbf = 32767

; LOAD SOUNDFONTS
gienginenum1 fluidEngine
gienginenum2 fluidEngine
isfnum1 fluidLoad "Piano Steinway Grand Model C (21,738KB).sf2", gienginenum1, 1
; Bright Steinway, program 1, channel 1
fluidProgramSelect gienginenum1, 1, isfnum1, 0, 1
; Concert Steinway with reverb, program 2, channel 3
fluidProgramSelect gienginenum1, 3, isfnum1, 0, 2
isfnum2 fluidLoad "63.3mg The Sound Site Album Bank V1.0.SF2", gienginenum2, 1
; General MIDI, program 50, channel 2
fluidProgramSelect gienginenum2, 2, isfnum2, 0, 50

; SEND NOTES TO STEINWAY SOUNDFONT

instr 1 ; FluidSynth Steinway Rev
; INITIALIZATION
mididefault 60, p3 ; Default duration of 60 -- overridden by score.
midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
ichannel = 1
ikey = p4
ivelocity = p5
```

```

    istatus      = 144
    fluidControl gienginenum1, istatus, ichannel, ikey, ivelocity
endin

instr 2 ; GM soundfont
; INITIALIZATION
    mididefault 60, p3 ; Default duration of 60 -- overridden by score.
    midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
    ichannel    = 2
    ikey        = p4
    ivelocity   = p5
    istatus     = 144
    fluidNote   gienginenum2, ichannel, ikey, ivelocity
endin

instr 3 ; FluidSynth Steinway Rev
; INITIALIZATION
    mididefault 60, p3 ; Default duration of 60 -- overridden by score.
    midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
    ichannel    = 3
    ikey        = p4
    ivelocity   = p5
    istatus     = 144
    fluidNote   gienginenum1, ichannel, ikey, ivelocity
endin

; COLLECT AUDIO FROM ALL SOUND FONTS

instr 100 ; Fluidsynth output
; INITIALIZATION
; Normalize so iamplitude for p5 of 80 == ampdb(80).
    iamplitude = ampdb(p5) * (10000.0 / 0.1)
; AUDIO
    aleft, aright fluidAllOut
    outs aleft * iamplitude, aright * iamplitude
endin

```

Voici un autre exemple plus complexe des opcodes fluidsynth écrit par Istvan Varga. Il utilise le fichier *fluidcomplex.csd* [examples/fluidcomplex.csd].

```

<CsSoundSynthesizer>
<CsOptions>
-d -m229 -o dac -T -F midifile.mid
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1

; Example by Istvan Varga

; disable triggering of instruments by MIDI events

ichn = 1
lp1:
    massign    ichn, 0
    loop_le    ichn, 1, 16, lp1
    pgmassign  0, 0

; initialize FluidSynth

gifld  fluidEngine
gisf2  fluidLoad "07AcousticGuitar.sf2", gifld, 1

; k-rate version of fluidProgramSelect

opcode fluidProgramSelect_k, 0, kkkkk
    keng, kchn, ksf2, kbnk, kpre xin
    igoto     skipInit
doInit:
    fluidProgramSelect i(keng), i(kchn), i(ksf2), i(kbnk), i(kpre)
    reinit      doInit
    rireturn
skipInit:

```

```

endop

instr 1
; initialize channels
kchn init 1
if (kchn == 1) then
lp2:
    fluidControl gifld, 192, kchn - 1, 0, 0
    fluidControl gifld, 176, kchn - 1, 7, 100
    fluidControl gifld, 176, kchn - 1, 10, 64
    loop_le kchn, 1, 16, lp2
endif

; send any MIDI events received to FluidSynth
nxt:
kst, kch, kd1, kd2 midin
if (kst != 0) then
    if (kst != 192) then
        fluidControl gifld, kst, kch - 1, kd1, kd2
    else
        fluidProgramSelect_k gifld, kch - 1, gisf2, 0, kd1
    endif
    kgoto nxt
endif

; get audio output from FluidSynth
aL, aR fluidOut gifld
    outs aL, aR
endin

</CsInstruments>
<CsScore>

i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*fluidEngine, fluidNote, fluidLoad*

## Crédits

Opcode par Michael Gogins (gogins@pipeline.com). Merci à Peter Hanappe pour Fluidsynth, et à Steven Yi pour avoir réalisé qu'il était nécessaire de diviser Fluidsynth en plusieurs opcodes Csound différents.



# fluidCCi

fluidCCi — Envoie un message de données de contrôleur MIDI à fluid.

## Syntaxe

```
fluidCCi iEngineNumber, iChannelNumber, iControllerNumber, iValue
```

## Description

Envoie un message de données de contrôleur MIDI (numéro du contrôleur MIDI et valeur à utiliser) à un moteur fluid spécifié par son numéro, sur le numéro de canal MIDI indiqué.

## Initialisation

*iEngineNumber* -- numéro du moteur affecté par fluidEngine

*iChannelNumber* -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

*iControllerNumber* -- numéro du contrôleur MIDI à utiliser pour ce message

*iValue* -- valeur à affecter au contrôleur (habituellement 0-127)

## Exécution

Cet opcode est utilisé pour affecter des valeurs de contrôleur pendant l'initialisation. Pour des changements continus, utilisez fluidCCK.

## Voir Aussi

*fluidEngine, fluidNote, fluidLoad, fluidCCK*

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

# fluidCCk

fluidCCk — Envoie un message de données de contrôleur MIDI à fluid.

## Syntaxe

```
fluidCCk iEngineNumber, iChannelNumber, iControllerNumber, kValue
```

## Description

Envoie un message de données de contrôleur MIDI (numéro du contrôleur MIDI et valeur à utiliser) à un moteur fluid spécifié par son numéro, sur le numéro de canal MIDI indiqué.

## Initialisation

*iEngineNumber* -- numéro du moteur affecté par fluidEngine

*iChannelNumber* -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

*iControllerNumber* -- numéro du contrôleur MIDI à utiliser pour ce message

## Exécution

*kValue* -- valeur à affecter au contrôleur (habituellement 0-127)

## Voir Aussi

*fluidEngine*, *fluidNote*, *fluidLoad*, *fluidCCi*

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

# fluidControl

fluidControl — Envoie un note on, un note off, et d'autres messages MIDI à un preset SoundFont.

## Syntaxe

```
fluidControl ienginenum, kstatus, kchannel, kdata1, kdata2
```

## Description

Les opcodes fluid fournissent une intégration simple dans des opcodes de Csound du synthétiseur Fluidsynth SoundFont2 de Peter Hanappe. Cette implémentation accepte les messages MIDI de note on, note off, de contrôleur, de pitch bend ou de changement de programme au taux-k. La polyphonie maximale est de 4096 voix simultanées. N'importe quel nombre de SoundFonts peuvent être chargés et joués simultanément.

## Initialisation

*ienginenum* -- numéro du moteur affecté par fluidEngine

## Exécution

*kstatus* -- octet d'état du message de canal MIDI : 128 pour note off, 144 pour note on, 176 pour control change, 192 for program change, ou 224 pour pitch bend.

*kchannel* -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

*kdata1* -- Pour note on, numéro de touche MIDI : de 0 (le plus bas) à 127 (le plus haut), où 60 est le do médian. Pour les messages de contrôleur continu, le numéro du contrôleur.

*kdata2* -- Pour note on, la vélocité de touche MIDI : de 0 (pas de son) à 127 (le plus fort). Pour les messages de contrôleur continu, la valeur du contrôleur.

Appelez fluidControl dans les définitions d'instrument qui jouent réellement des notes et qui envoient des messages de contrôle. Chaque définition d'instrument doit utiliser de manière cohérente un canal MIDI qui a été affecté à un programme Fluidsynth au moyen de fluidLoad.

Dans cette implémentation, les effets SoundFont tels que chorus ou réverbération sont utilisés si et seulement s'ils sont présents par défaut pour le preset. Il n'y a aucun moyen d'activer ou d'arrêter de tels effets, ou de changer leurs paramètres, depuis Csound.

## Voir Aussi

*fluidEngine, fluidNote, fluidLoad*

## Crédits

Opcodes par Michael Gogins (gogins@pipeline.com). Merci à Peter Hanappe pour Fluidsynth, et à Steven Yi pour avoir réalisé qu'il était nécessaire de diviser Fluidsynth en plusieurs opcodes Csound différents.

Nouveau dans Csound5.00

# fluidEngine

fluidEngine — Crée une instance de moteur fluidsynth.

## Syntaxe

```
ienginenum fluidEngine [iReverbEnabled] [, iChorusEnabled] [,iNumChannels] [, iPolyphony]
```

## Description

Crée une instance de moteur fluidsynth, et retourne *ienginenum* pour identifier le moteur. *ienginenum* est passé à d'autres opcodes pour charger et jouer des SoundFonts et pour assembler le son généré.

## Initialisation

*ienginenum* -- numéro du moteur affecté par fluidEngine

*iReverbEnabled* -- fixé de manière facultative à 0 pour désactiver d'éventuels effets de réverbération dans les SoundFonts chargés.

*iChorusEnabled* -- fixé de manière facultative à 0 pour désactiver d'éventuels effets de chorus dans les SoundFonts chargés.

*iNumChannels* -- nombre de canaux à utiliser ; de 16 à 256, la valeur par défaut de Csound est 256 (la valeur par défaut de Fluidsynth est 16).

*iPolyphony* -- nombre de voix à jouer en parallèle ; de 16 à 4096, la valeur par défaut de Csound est 4096 (la valeur par défaut de Fluidsynth est 256). Note : ce n'est pas le nombre de notes jouées simultanément car une seule note peut utiliser plusieurs voix en fonction des zones d'instrument et de la vélocité et/ou du numéro de touche de la note jouée.

## Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluidAllOut.orc* [examples/fluidAllOut.orc].

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
odbfs = 32767

; LOAD SOUNDFONTS
gienginenum1 fluidEngine
gienginenum2 fluidEngine
isfnum1 fluidLoad "Piano Steinway Grand Model C (21,738KB).sf2", gienginenum1, 1
; Bright Steinway, program 1, channel 1
fluidProgramSelect gienginenum1, 1, isfnum1, 0, 1
; Concert Steinway with reverb, program 2, channel 3
fluidProgramSelect gienginenum1, 3, isfnum1, 0, 2
isfnum2 fluidLoad "63.3mg The Sound Site Album Bank V1.0.SF2", gienginenum2, 1
; General MIDI, program 50, channel 2
fluidProgramSelect gienginenum2, 2, isfnum2, 0, 50

; SEND NOTES TO STEINWAY SOUNDFONT
instr 1 ; FluidSynth Steinway Rev
; INITIALIZATION
mididefault 60, p3 ; Default duration of 60 -- overridden by score.
midinoteonkey p4, p5 ; Channels MIDI input to pfields.
```

```

; Use channel assigned in fluidload.
ichannel = 1
ikey = p4
ivelocity = p5
istatus = 144
fluidControl gienginenum1, istatus, ichannel, ikey, ivelocity
endin

instr 2 ; GM soundfont
; INITIALIZATION
mididefault 60, p3 ; Default duration of 60 -- overridden by score.
midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
ichannel = 2
ikey = p4
ivelocity = p5
istatus = 144
fluidNote gienginenum2, ichannel, ikey, ivelocity
endin

instr 3 ; FluidSynth Steinway Rev
; INITIALIZATION
mididefault 60, p3 ; Default duration of 60 -- overridden by score.
midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
ichannel = 3
ikey = p4
ivelocity = p5
istatus = 144
fluidNote gienginenum1, ichannel, ikey, ivelocity
endin

; COLLECT AUDIO FROM ALL SOUNDFONTS

instr 100 ; Fluidsynth output
; INITIALIZATION
; Normalize so iamplitude for p5 of 80 == ampdb(80).
iamplitude = ampdb(p5) * (10000.0 / 0.1)
; AUDIO
aleft, aright fluidAllOut
outs aleft * iamplitude, aright * iamplitude
endin

```

Voici un exemple des opcodes fluidsynth qui fait appel à 2 moteurs. Il utilise le fichier *fluid-2.orc* [examples/fluid-2.orc].

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 32767

; LOAD SOUNDFONTS
gienginenum1 fluidEngine
gienginenum2 fluidEngine
isfnum1 fluidLoad "Piano Steinway Grand Model C (21,738KB).sf2", gienginenum1, 1
; Bright Steinway, program 1, channel 1
fluidProgramSelect gienginenum1, 1, isfnum1, 0, 1
; Concert Steinway with reverb, program 2, channel 3
fluidProgramSelect gienginenum1, 3, isfnum1, 0, 2
isfnum2 fluidLoad "63.3mg The Sound Site Album Bank V1.0.SF2", gienginenum2, 1
; General MIDI, program 50, channel 2
fluidProgramSelect gienginenum2, 2, isfnum2, 0, 50

; SEND NOTES TO STEINWAY SOUNDFONT

instr 1 ; FluidSynth Steinway Rev
; INITIALIZATION
mididefault 60, p3 ; Default duration of 60 -- overridden by score.
midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
ichannel = 1
ikey = p4
ivelocity = p5
fluidNote gienginenum1, ichannel, ikey, ivelocity
endin

```

```

instr 2 ; GM soundfont
; INITIALIZATION
    mididefault 60, p3 ; Default duration of 60 -- overridden by score.
    midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
    ichannel = 2
    ikey = p4
    ivelocity = p5
    fluidNote gienginenum2, ichannel, ikey, ivelocity
endin

instr 3 ; FluidSynth Steinway Rev
; INITIALIZATION
    mididefault 60, p3 ; Default duration of 60 -- overridden by score.
    midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
    ichannel = 3
    ikey = p4
    ivelocity = p5
    fluidNote gienginenum1, ichannel, ikey, ivelocity
endin

; COLLECT AUDIO FROM ALL SOUND FONTS

instr 100 ; Fluidsynth output
; INITIALIZATION
; Normalize so iamplitude for p5 of 80 == ampdb(80).
    iamplitude1 = ampdb(p5) * (10000.0 / 0.1)
    iamplitude2 = ampdb(p6) * (10000.0 / 0.1)

; AUDIO
    aleft1, aright1 fluidOut gienginenum1
    aleft2, aright2 fluidOut gienginenum2
    outs (aleft1 * iamplitude1) + (aleft2 * iamplitude2), \
        (aright1 * iamplitude1) + (aright2 * iamplitude2)
endin

```

Voici un autre exemple plus complexe des opcodes fluidsynth écrit par Istvan Varga. Il utilise le fichier *fluidcomplex.csd* [examples/fluidcomplex.csd].

```

<CsoundSynthesizer>
<CsOptions>
-d -m229 -o dac -T -F midifile.mid
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1

; Example by Istvan Varga

; disable triggering of instruments by MIDI events

ichn = 1
lp1:
    massign ichn, 0
    loop_le ichn, 1, 16, lp1
    pgmassign 0, 0

; initialize FluidSynth

gifld fluidEngine
gisf2 fluidLoad "07AcousticGuitar.sf2", gifld, 1

; k-rate version of fluidProgramSelect

opcode fluidProgramSelect_k, 0, kkkkk
    keng, kchn, ksf2, kbnk, kpre xin
    igoto skipInit
doInit:
    fluidProgramSelect i(keng), i(kchn), i(ksf2), i(kbnk), i(kpre)
    reinit doInit
    rireturn
skipInit:
endop

instr 1

```

```

; initialize channels
kchn init 1
if (kchn == 1) then
lp2:
    fluidControl gifld, 192, kchn - 1, 0, 0
    fluidControl gifld, 176, kchn - 1, 7, 100
    fluidControl gifld, 176, kchn - 1, 10, 64
    loop_le kchn, 1, 16, lp2
endif

; send any MIDI events received to FluidSynth
nxt:
kst, kch, kd1, kd2 midin
if (kst != 0) then
    if (kst != 192) then
        fluidControl gifld, kst, kch - 1, kd1, kd2
    else
        fluidProgramSelect_k gifld, kch - 1, gisf2, 0, kd1
    endif
    kgoto nxt
endif

; get audio output from FluidSynth
aL, aR fluidOut gifld
    outs aL, aR
endin

</CsInstruments>
<CsScore>

i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*fluidNote, fluidLoad*

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Les paramètres facultatifs *iNumChannels* et *iPolyphony* ont été ajoutés dans la version 5.07



# fluidLoad

**fluidLoad** — Charge un SoundFont dans un fluidEngine, en listant éventuellement le contenu du SoundFont.

## Syntaxe

```
isfnum fluidLoad soundfont, ienginenum[, ilistpresets]
```

## Description

Charge un SoundFont dans une instance d'un fluidEngine, en listant éventuellement les banques et les presets du SoundFont.

## Initialisation

*isfnum* -- numéro affecté au soundfont qui vient d'être chargé.

*soundfont* -- chaîne spécifiant le nom de fichier d'un SoundFont. Notez que n'importe quel nombre de SoundFonts peuvent être chargés (évidemment, par différents appels de fluidLoad).

*ienginenum* -- numéro du moteur affecté par fluidEngine

*ilistpresets* -- facultatif, s'il est spécifié, tous les programmes Fluidsynth du SoundFont qui vient d'être chargé sont listés. Un programme FluidSynth est une combinaison d'ID de SoundFont, de numéro de banque, et de numéro de preset qui est affecté à un canal MIDI.

## Exécution

Appelez fluidLoad dans l'en-tête de l'orchestre, autant de fois que vous voulez. Le même SoundFont peut être appelé pour affecter des programmes à des canaux MIDI autant de fois que l'on veut ; le SoundFont n'est chargé que la première fois.

## Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluid.orc* [examples/fluid.orc].

```
sr = 44100
ksmps = 100
nchnls = 2

giengine fluidEngine
isfnum fluidLoad "07AcousticGuitar.sf2", giengine, 1
fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1
    mididefault      60, p3
    midinoteonkey    p4, p5

    ikey init        p4
    ivel init        p5

    fluidNote        giengine, 1, ikey, ivel
endin

instr 99
    imvol init       70000
```

```
asigl, asigr fluidOut giengine  
outs asigl * imvol, asigr * imvol  
endin
```

Voir *fluidEngine* pour plus d'exemples.

## Voir Aussi

*fluidEngine*, *fluidNote*

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Nouveau dans Csound5.00

# fluidNote

fluidNote — Joue une note sur un canal dans un moteur fluidsynth.

## Syntaxe

```
fluidNote iengineum, ichannelnum, imidikey, imidivel
```

## Description

Joue une note de hauteur *imidikey* et de vélocité *imidivel* sur le canal *ichannelnum* du fluidEngine numé-  
ro *iengineum*.

## Initialisation

*iengineum* -- numéro du moteur affecté par fluidEngine

*ichannelnum* -- numéro de canal sur lequel jouer la note dans le fluidEngine donné

*imidikey* -- touche MIDI de la note (0-127)

*imidivel* -- vélocité MIDI de la note (0-127)

## Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluid.orc* [examples/fluid.orc].

```
sr = 44100
ksmps = 100
nchnls = 2

giengine fluidEngine
isfnum fluidLoad "07AcousticGuitar.sf2", giengine, 1
fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1
    mididefault 60, p3
    midinoteonkey p4, p5

    ikey init p4
    ivel init p5

    fluidNote giengine, 1, ikey, ivel
endin

instr 99
    imvol init 70000
    asigl, asigr fluidOut giengine
    outs asigl * imvol, asigr * imvol
endin
```

Voir *fluidEngine* pour plus d'exemples.

## Voir Aussi

*fluidEngine*, *fluidLoad*

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

# fluidOut

fluidOut — Envoie en sortie le son d'un fluidEngine donné.

## Syntaxe

```
aleft, aright fluidOut ienginenum
```

## Description

Envoie en sortie le son d'un fluidEngine donné.

## Initialisation

*ienginenum* -- numéro du moteur affecté par fluidEngine

## Exécution

*aleft* -- Canal de sortie audio gauche.

*aright* -- Canal de sortie audio droite.

Appelez fluidOut dans une définition d'instrument dont le numéro est supérieur à ceux de toutes les définitions d'instrument de contrôle de fluid. Tous les SoundFonts utilisés par le fluidEngine numéro *ienginenum* envoient leur sortie audio à cet opcode. Envoyez une note de durée indéterminée à cet instrument afin d'activer les SoundFonts pour une durée suffisante.

## Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluid.orc* [examples/fluid.orc].

```
sr = 44100
ksmps = 100
nchnls = 2

giengine fluidEngine
isfnum fluidLoad "07AcousticGuitar.sf2", giengine, 1
fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1
  mididefault      60, p3
  midinoteonkey    p4, p5

  ikey init        p4
  ivel init        p5

  fluidNote        giengine, 1, ikey, ivel
endin

instr 99
  imvol init       70000
  asigl, asigr fluidOut giengine
  outs            asigl * imvol, asigr * imvol
endin
```

Voir *fluidEngine* pour plus d'exemples.

## Voir Aussi

*fluidEngine, fluidNote, fluidLoad*

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Nouveau dans Csound5.00

# fluidProgramSelect

fluidProgramSelect — Affecte un preset d'un SoundFont à un canal d'un fluidEngine.

## Syntaxe

```
fluidProgramSelect ienginenum, ichannelnum, isfnum, ibanknum, ipresetnum
```

## Description

Affecte un preset d'un SoundFont à un canal d'un fluidEngine.

## Initialisation

*ienginenum* -- numéro du moteur affecté par fluidEngine

*ichannelnum* -- numéro du canal auquel affecter le preset dans le fluidEngine donné

*isfnum* -- numéro du SoundFont duquel le preset est issu

*ibanknum* -- numéro de la banque dans le SoundFont de laquelle le preset est issu

*ipresetnum* -- numéro du preset à affecter

## Exemples

Voici un exemple des opcodes fluidsynth. Il utilise le fichier *fluid.orc* [examples/fluid.orc].

```
sr = 44100
ksmps = 100
nchnls = 2

giengine fluidEngine
isfnum   fluidLoad "07AcousticGuitar.sf2", giengine, 1
         fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1
    mididefault    60, p3
    midinoteonkey  p4, p5

    ikey   init    p4
    ivel   init    p5

    fluidNote    giengine, 1, ikey, ivel
endin

instr 99
    imvol   init    70000
    asigl, asigr fluidOut giengine
    outs    asigl * imvol, asigr * imvol
endin
```

Voir *fluidEngine* pour plus d'exemples.

## Voir Aussi

*fluidEngine*, *fluidNote*, *fluidLoad*

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.



# fluidSetInterpMethod

fluidSetInterpMethod — Fixe la méthode d'interpolation pour un canal dans le moteur fluidsynth.

## Syntax

```
fluidSetInterpMethod ienginenum, ichannelnum, iInterpMethod
```

## Description

Fixe la méthode d'interpolation pour un canal dans le moteur fluidsynth. Les méthodes d'interpolation d'ordre inférieur donnent une restitution plus rapide et de moindre qualité tandis que les méthodes d'interpolation d'ordre élevé donnent une restitution plus lente et de meilleure qualité. L'interpolation par défaut pour un canal est du quatrième ordre.

## Initialisation

*ienginenum* -- numéro du moteur alloué par *fluidEngine*

*ichannelnum* -- numéro de canal à utiliser pour le preset dans le moteur fluidsynth donné

*iInterpMethod* -- méthode d'interpolation, l'une des suivantes

- 0 -- Pas d'interpolation
- 1 -- Interpolation linéaire
- 4 -- Interpolation d'ordre 4 (par défaut)
- 7 -- Interpolation d'ordre 7 (la plus haute)

## Voir Aussi

*fluidEngine*

## Crédits

Auteur : Steven Yi

Nouveau dans la version 5.07

# FLvalue

FLvalue — Montre la valeur courante d'un valuateur FLTK.

## Description

*FLvalue* montre la valeur courante d'un valuateur dans un champ texte.

## Syntaxe

```
ihandle FLvalue "label", iwidth, iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le valuateur correspondant. Il peut être utilisé comme argument *idisp* d'un valuateur.

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*FLvalue* montre la valeur courante d'un valuateur dans un champ texte. Il retourne *ihandle* qui peut être utilisé comme argument *idisp* d'un valuateur (voir la section *Valuateurs FLTK*). De cette manière, les valeurs de ce valuateur seront montrées dynamiquement dans un champ texte.



### Note

Noter que *FLvalue* n'est pas un valuateur et que sa valeur ne peut pas être modifiée. La valeur d'un widget *FLvalue* ne doit être fixée que par d'autres widgets, et PAS depuis *FLsetVal* ou *FLsetVal\_i* car cela pourrait planter Csound.

## Exemples

Voici un exemple de l'opcode FLvalue. Il utilise le fichier *FLvalue.csd* [exemples/FLvalue.csd].

### Exemple 184. Exemple de l'opcode FLvalue.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLvalue.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Using the opcode flvalue to display the output of a slider
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Value Display Box", 900, 200, 50, 50
; Width of the value display box in pixels
iwidth = 50
; Height of the value display box in pixels
iheight = 20
; Distance of the left edge of the value display
; box from the left edge of the panel
ix = 65
; Distance of the top edge of the value display
; box from the top edge of the panel
iy = 55

idispatch FLvalue "Hertz", iwidth, iheight, ix, iy
gkfreq, ihandle FLslider "Frequency", 200, 5000, -1, 5, idispatch, 750, 30, 125, 50
FLsetVal i 500, ihandle
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
iamp = 15000
ifn = 1
asig oscili iamp, gkfreq, ifn
out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLbox, FLbutBank, FLbutton, FLprintk, FLprintk2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLvkeybd

FLvkeybd — Un opcode de widget FLTK qui crée un widget de clavier virtuel.

## Description

Un opcode de widget FLTK qui crée un widget de clavier virtuel. Il doit être utilisé avec le pilote du clavier virtuel midi pour opérer correctement. Cet opcode est utile pour faire des versions de démonstration d'orchestres MIDI avec le clavier virtuel inclus dans la fenêtre principale.



### Note

La version widget du clavier virtuel ne comprend pas les réglettes MIDI que l'on trouve dans la version complète de la fenêtre du clavier virtuel.

## Syntaxe

**FLvkeybd** "keyboard.map", *iwidth*, *iheight*, *ix*, *iy*

## Initialisation

« *keyboard.map* » -- une chaîne de caractères entre guillemets contenant le mappage du clavier à utiliser. On peut fournir une chaîne vide ("" ) pour utiliser les valeurs de nom de banque/canal par défaut. Voir Clavier Virtuel Midi pour plus d'information sur les mappages du clavier.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix*-- position horizontale du coin supérieur gauche du clavier, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy*-- position verticale du coin supérieur gauche du clavier, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).



### Note

La largeur et la hauteur standard du clavier virtuel est de 624x120 pour la version dialogue qui est montrée quand *FLvkeybd* n'est pas utilisé.

## Voir Aussi

*FLbutton*, *FLbox*, *FLbutBank*, *FLprintk*, *FLprintk2*, *FLvalue*

## Crédits

Auteur : Steven Yi

Nouveau dans la version 5.05

# FLvslidBnk

FLvslidBnk — Un widget FLTK contenant un banc de réglettes verticales.

## Description

*FLvslidBnk* est un widget contenant un banc de réglettes verticales.

## Syntaxe

```
FLvslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
    [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]
```

## Initialisation

« *names* » -- une chaîne de caractères entre guillemets contenant le nom de chaque réglette. Chaque réglette peut avoir un nom différent. Chaque nom est séparé par un caractère « @ », par exemple : « fréquence@amplitude@coupure ». Il est possible de ne fournir aucun nom en donnant un seul espace « ». Dans ce cas, l'opcode affectera automatiquement un numéro en progression ascendante comme étiquette pour chaque réglette.

*inumsliders* -- le nombre de réglettes.

*ioutable* (facultatif, 0 par défaut) -- numéro d'une table allouée préalablement dans laquelle seront stockées les valeurs de sortie de chaque réglette. Il faut s'assurer que la taille de la table est suffisante pour contenir toutes les cellules de sortie, sinon Csound plantera avec une erreur de segmentation. En affectant zéro à cet argument, la sortie sera dirigée vers l'espace zak dans la zone de taux-k. Dans ce cas, l'espace zak doit avoir été alloué au préalable avec l'opcode *zakinit* et il faut s'assurer que la taille d'allocation est suffisante pour couvrir toutes les réglettes. La valeur par défaut est zéro (c'est-à-dire stockage de la sortie dans l'espace zak).

*istart\_index* (facultatif, 0 par défaut) -- un nombre entier indiquant un décalage des positions des cellules de sortie. Il peut être positif pour permettre l'allocation en sortie de plusieurs bancs de réglettes dans la même table ou dans l'espace zak. La valeur par défaut est zéro (pas de décalage).

*iminmaxtable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste de couples min-max pour chaque réglette. Une valeur de zéro signifie l'intervalle allant de 0 à 1 pour toutes les réglettes, sans fournir de table. La valeur par défaut est zéro.

*iexptable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste d'identifiants (des nombres entiers) fournis pour modifier le comportement de chaque réglette de manière indépendante. Les identifiants peuvent avoir les valeurs suivantes :

- -1 -- courbe de réponse exponentielle
- 0 -- réponse linéaire
- nombre > 0 -- suit la courbe d'une table définie au préalable pour mettre en forme la réponse de la réglette correspondante. Dans ce cas, ce nombre correspond au numéro de la table.

On peut souhaiter que toutes les réglettes du banc aient la même courbe de réponse (exponentielle ou linéaire). Dans ce cas, on peut affecter -1 ou 0 à *iexptable* sans se préoccuper de définir auparavant une table. La valeur par défaut est zéro (toutes les réglettes ont une réponse linéaire sans avoir à fournir de

table).

*ityetable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste d'identifiants (des nombres entiers) fournis pour modifier l'aspect de chaque réglette de manière indépendante. Les identifiants peuvent avoir les valeurs suivantes :

- 0 = Réglette stylée
- 1 = Réglette pleine
- 3 = Réglette normale
- 5 = Réglette stylée
- 7 = Réglette stylée en creux

On peut souhaiter que toutes les réglettes du banc aient le même aspect. Dans ce cas, on peut affecter un nombre négatif à *ityetable* sans se préoccuper de définir auparavant une table. Les nombres négatifs ont la même signification que les identifiants positifs correspondants sauf que le même aspect est affecté à toutes les réglettes. On peut aussi donner un aspect aléatoire à chaque réglette en affectant à *ityetable* un nombre négatif inférieur à -7. La valeur par défaut est 0 (toutes les réglettes sont stylées, sans avoir à fournir de table).

On peut ajouter 20 à une valeur dans la table pour donner l'aspect "plastique" à la réglette, ou soustraire 20 si l'on veut affecter la valeur à tous les widgets sans définir une table (par exemple -21 pour donner à toutes les réglettes le type Plastique Plein).

*iwidth* (facultatif) -- largeur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées sous cette zone.

*iheight* (facultatif) -- hauteur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées sous cette zone.

*ix* (facultatif) -- position horizontale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace en-dessous de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*iy* (facultatif) -- position verticale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace en-dessous de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLvslidBnk* est un widget contenant un banc de réglettes verticales. On peut y mettre n'importe quel nombre de réglettes (argument *inumsliders*). La sortie de toutes les réglettes est stockée dans une table allouée au préalable ou dans l'espace zak (argument *ioutable*). Il est possible de déterminer la première position de la table (ou de l'espace zak) dans lequel stocker la sortie de la première réglette au moyen de l'argument *istart\_index*.

Chaque réglette peut avoir une étiquette individuelle placée sous elle. Les étiquettes sont définies par l'argument « *names* ». L'intervalle de sortie de chaque réglette peut être fixé individuellement au moyen d'une table externe (argument *iminmaxtable*). La courbe de réponse de chaque réglette peut être fixée individuellement, au moyen d'une liste d'identifiants placés dans une table (argument *ixptable*). Il est pos-

sible de définir l'aspect de chaque réglette indépendamment ou de donner le même aspect à toutes les réglettes (argument *ityetable*).

Les arguments *iwidth*, *iheight*, *ix* et *iy* déterminent la largeur, la hauteur, les positions horizontale et verticale de la zone rectangulaire contenant les réglettes. Noter que l'étiquette de chaque réglette est placée en-dessous et n'est pas incluse dans la zone rectangulaire contenant les réglettes. Ainsi l'utilisateur doit laisser assez d'espace sous le banc en affectant une valeur suffisante à *iy* afin que les étiquettes soient visibles.

*FLvslidBnk* est identique à *FLslidBnk* sauf qu'il contient des réglettes verticales plutôt qu'horizontales. Comme la largeur de chaque réglette est souvent petite, il est recommandé de ne laisser qu'un seul espace dans la chaîne de noms (" "), ce qui fait que chaque réglette sera numérotée automatiquement.



## IMPORTANT !

Noter que les tables utilisées par *FLvslidBnk* doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

## Exemples

Voici un exemple de l'opode *FLvslidBnk*. Il utilise le fichier *FLvslidBnk.csd* [examples/FLvslidBnk.csd].

### Exemple 185. Exemple de l'opode *FLvslidBnk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

gityetable ftgen 0, 0, 8, -2, 1, 1, 3, 3, 5, 5, 7, 7
giouttable ftgen 0, 0, 8, -2, 0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1

FLpanel "Slider Bank", 400, 400, 50, 50
;Number of sliders
inum = 8
; Table to store output
iouttable = giouttable
; Width of the slider bank in pixels
iwidth = 350
; Height of the slider in pixels
iheight = 160
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 30
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 10
; Table containing fader types
ityetable = gityetable
FLvslidBnk "1@2@3@4@5@6@7@8@9@10@11@12@13@14@15@16", 16 , iouttable , iwidth , iheight , ix \
```

```
        , iy , itypetable
        FLvslidBnk " ", inum , iouttable , iwidth , iheight , ix \
        , iy + 200 , -23
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
;Dummy instrument
endin

</CsInstruments>
<CsScore>

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*FLslider, FLslidBnk*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06



# FLvslidBnk2

FLvslidBnk2 — Un widget FLTK contenant un banc de réglottes verticales.

## Description

*FLvslidBnk2* est un widget contenant un banc de réglottes verticales.

## Syntaxe

**FLvslidBnk2** "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart\_index]

## Initialisation

« *names* » -- une chaîne de caractères entre guillemets contenant le nom de chaque réglotte. Chaque réglotte peut avoir un nom différent. Chaque nom est séparé par un caractère « @ », par exemple : « fréquence@amplitude@coupure ». Il est possible de ne fournir aucun nom en donnant un seul espace « ». Dans ce cas, l'opcode affectera automatiquement un numéro en progression ascendante comme étiquette pour chaque réglotte.

*inumsliders* -- le nombre de réglottes.

*ioutable* (facultatif, 0 par défaut) -- numéro d'une table allouée préalablement dans laquelle seront stockées les valeurs de sortie de chaque réglotte. Il faut s'assurer que la taille de la table est suffisante pour contenir toutes les cellules de sortie, sinon Csound plantera avec une erreur de segmentation. En affectant zéro à cet argument, la sortie sera dirigée vers l'espace zak dans la zone de taux-k. Dans ce cas, l'espace zak doit avoir été alloué au préalable avec l'opcode *zakinit* et il faut s'assurer que la taille d'allocation est suffisante pour couvrir toutes les réglottes. La valeur par défaut est zéro (c'est-à-dire stockage de la sortie dans l'espace zak).

*iconfigtable* -- dans les opcodes *FLslidBnk2* et *FLvslidBnk2*, cette table remplace *iminmaxtable*, *iexp-table* et *istyletable*, tous ces paramètres étant placés dans une seule table. Cette table doit être remplie avec un groupe de quatre paramètres pour chaque réglotte de la façon suivante :

min1, max1, exp1, style1, min2, max2, exp2, style2, min3, max3, exp3, style3 etc.

par exemple en utilisant GEN02 on peut taper :

*inumftgen* 1,0,256, -2, 0,1,0,1, 100, 5000, -1, 3, 50, 200, -1, 5,..... [etc]

Dans cet exemple la première réglotte reçoit les paramètres [0, 1, 0, 1] (valeurs comprises entre 0 et 1, réponse linéaire, aspect réglotte pleine), la seconde réglotte reçoit les paramètres [100, 5000, -1, 3] (valeurs comprises entre 100 et 5000, réponse exponentielle, aspect réglotte normale), la troisième réglotte reçoit les paramètres [50, 200, -1, 5] (valeurs comprises entre 50 et 200, réponse exponentielle, aspect réglotte stylée), et ainsi de suite.

*iwidth* (facultatif) -- largeur de la zone rectangulaire contenant toutes les réglottes du banc, à l'exclusion des étiquettes qui sont placées sous cette zone.

*iheight* (facultatif) -- hauteur de la zone rectangulaire contenant toutes les réglottes du banc, à l'exclusion des étiquettes qui sont placées sous cette zone.

*ix* (facultatif) -- position horizontale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglottes appartenant au banc. Il faut laisser suffisamment d'espace en-dessous de ce rectangle afin que les étiquettes des réglottes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*iy* (facultatif) -- position verticale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglottes appartenant au banc. Il faut laisser suffisamment d'espace en-dessous de ce rectangle afin que les étiquettes des réglottes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*istart\_index* (facultatif, 0 par défaut) -- un nombre entier indiquant un décalage des positions des cellules de sortie. Il peut être positif pour permettre l'allocation en sortie de plusieurs bancs de réglottes dans la même table ou dans l'espace zak. La valeur par défaut est zéro (pas de décalage).

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLvslidBnk2* est un widget contenant un banc de réglottes verticales. On peut y mettre n'importe quel nombre de réglottes (argument *inumsliders*). La sortie de toutes les réglottes est stockée dans une table allouée au préalable ou dans l'espace zak (argument *ioutable*). Il est possible de déterminer la première position de la table (ou de l'espace zak) dans laquelle stocker la sortie de la première réglotte au moyen de l'argument *istart\_index*.

Chaque réglotte peut avoir une étiquette individuelle placée sous elle. Les étiquettes sont définies par l'argument « *names* ». L'intervalle de sortie de chaque réglotte peut être fixé individuellement au moyen des valeurs *min* et *max* dans la table *iconfigtable*. La courbe de réponse de chaque réglotte peut être fixée individuellement, au moyen d'une liste d'identifiants placés dans la table *iconfigtable* (argument *exp*). Il est possible de définir l'aspect de chaque réglotte indépendamment ou de donner le même aspect à toutes les réglottes (argument *style* dans la table *iconfigtable*).

Les arguments *iwidth*, *iheight*, *ix* et *iy* déterminent la largeur, la hauteur, les positions horizontale et verticale de la zone rectangulaire contenant les réglottes. Noter que l'étiquette de chaque réglotte est placée en-dessous d'elle et n'est pas incluse dans la zone rectangulaire contenant les réglottes. Ainsi l'utilisateur doit laisser assez d'espace à la gauche du banc en affectant une valeur suffisante à *iy* afin que les étiquettes soient visibles.

*FLvslidBnk2* est identique à *FLslidBnk2* sauf qu'il contient des réglottes verticales plutôt qu'horizontales. Comme la largeur de chaque réglotte est souvent petite, il est recommandé de ne laisser qu'un seul espace dans la chaîne de noms (" "), ce qui fait que chaque réglotte sera numérotée automatiquement.



### IMPORTANT !

Noter que les tables utilisées par *FLvslidBnk2* doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

## Voir Aussi

*FLslider*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLxyin

FLxyin — Détecte la position de curseur de la souris dans une zone définie à l'intérieur d'un FLpanel.

## Description

Semblable à *xyin*, détecte la position de curseur de la souris dans une zone définie à l'intérieur d'un FL-panel.

## Syntaxe

```
koutx, kouty, kinside FLxyin ioutx_min, ioutx_max, iouty_min, iouty_max, \  
iwindx_min, iwindx_max, iwindy_min, iwindy_max [, iexpx, iexpy, ioutx, iouty]
```

## Initialisation

*ioutx\_min, ioutx\_max* - les valeurs limites de l'intervalle de sortie (X ou axe horizontal).

*iouty\_min, iouty\_max* - les valeurs limites de l'intervalle de sortie (Y ou axe vertical).

*iwindx\_min, iwindx\_max* - les coordonnées X des bords horizontaux de la zone sensible, relatives au FL-panel, en pixels.

*iwindy\_min, iwindy\_max* - les coordonnées Y des bords verticaux de la zone sensible, relatives au FL-panel, en pixels.

*iexpx, iexpy* - (facultatif) nombres entiers définissant le comportement des sorties x ou y : 0 -> la sortie est linéaire ; 1 -> la sortie est exponentielle ; tout autre nombre indique le numéro d'une table existante utilisée pour l'indexation. Noter que dans les opérations normales, la table doit être normalisée et unipolaire (tous les éléments de la table doivent être compris entre zéro et un). Dans ce cas, tous les éléments de la table seront mis à l'échelle en fonction de *imin* et de *imax*. Il est tout de même possible d'utiliser des tables non normalisées (créées avec un numéro de table négatif, qui peuvent contenir des éléments de n'importe quelle valeur), afin d'accéder aux valeurs courantes des éléments de la table, sans mise à l'échelle, en affectant 0 à *iout\_min* et 1 à *iout\_max*.

*ioutx, iouty* - (facultatif) valeurs de sortie initiales.

## Exécution

*koutx, kouty* - valeurs de sorties, mises à l'échelle selon les choix de l'utilisateur.

*kinside* - un drapeau indiquant si le curseur de la souris se trouve en dehors du rectangle de la zone définie. S'il est en dehors de la zone, *kinside* vaut zéro.

*FLxyin* détecte la position du curseur de la souris dans une zone définie à l'intérieur d'un *FLpanel*. Quand *FLxyin* est appelé, la position de la souris dans la zone choisie est retournée au taux-k. Il est possible de définir la zone sensible, ainsi que les valeurs minimale et maximale correspondant aux positions minimale et maximale de la souris. Il n'est pas nécessaire que les boutons de la souris soient appuyés pour que *FLxyin* fonctionne. Il est capable d'opérer correctement même si d'autres widgets (présents dans le *FLpanel*) chevauchent la zone sensible.

A l'inverse de la plupart des autres opcodes FLTK, *FLxyin* ne peut pas être utilisé dans l'en-tête, car ce n'est pas un widget. Ce n'est que la définition d'une zone de détection de la souris à l'intérieur d'un pan-

neau FLTK.

## Exemples

Voici un exemple de l'opcode FLxyin. Il utilise le fichier *FLxyin.csd* [examples/FLxyin.csd].

### Exemple 186. Exemple de l'opcode FLxyin.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=48000
ksmps=128
nchnls=2

; Example by Andres Cabrera 2007

FLpanel "FLxyin", 200, 100, -1, -1, 3
FLpanelEnd
FLrun

instr 1
  koutx, kouty, kinside FLxyin 0, 10, 100, 1000, 10, 190, 10, 90
  aout buzz 10000, kouty, koutx, 1
  printk2 koutx
  outs aout, aout
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 3600

e

</CsScore>
</CsoundSynthesizer>
```

Voici une autre exemple de l'opcode FLxyin. Il utilise le fichier *FLxyin-2.csd* [examples/FLxyin-2.csd].

### Exemple 187. Exemple de l'opcode FLxyin.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
kr=441
ksmps=100
nchnls=2

; Example by Gabriel Maldonado
```

```

    FLpanel "Move the mouse inside this panel to hear the effect",400,400
    FLpanel_end
    FLrun

    instr 1
k1, k2, kinside FLxyin 50, 1000, 50, 1000, 100, 300, 50, 250, -2,-3
; if k1 <= 50 || k1 >=5000 || k2 <=100 || k2 >= 8000 kgoto end ; if cursor is outside bounds, then don't
a1 oscili 3000, k1, 1
a2 oscili 3000, k2, 1

    outs a1,a2
    printk2 k1
    printk2 k2, 10
    printk2 kinside, 20
end:
    endin

</CsInstruments>
<CsScore>

f1 0 1024 10 1
f2 0 17 19 1 1 90 1
f3 0 17 19 2 1 90 1
i1 0 3600

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*FLpanel*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# fmb3

fmb3 — Utilise la synthèse FM pour créer un son d'orgue Hammond B3.

## Description

Utilise la synthèse FM pour créer un son d'orgue Hammond B3. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \  
      ifn4, ivfn
```

## Initialisation

*fmb3* prend 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 4

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode `fmb3`. Il utilise le fichier `fmb3.csd` [examples/fmb3.csd].

### Exemple 188. Exemple de l'opcode `fmb3`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmb3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 15000
  kfreq = 440
  kc1 = 5
  kc2 = 5
  kvdepth = 0.005
  kvrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 1
  ivfn = 1

  a1 fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate, \
      ifn1, ifn2, ifn3, ifn4, ivfn
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*fmbell, fmmetal, fmpercfl, fmrhode, fmwurlie*

## Crédits

Auteur : John ffitich (d'après Perry Cook)  
University of Bath, Codemist Ltd.



Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# fmbell

fmbell — Utilise la synthèse FM pour créer un son de cloche tube.

## Description

Utilise la synthèse FM pour créer un son de cloche tube. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \  
      ifn4, ivfn
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode `fmbell`. Il utilise le fichier `fmbell.csd` [examples/fmbell.csd].

### Exemple 189. Exemple de l'opcode `fmbell`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o fmbell.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 10000
  kfreq = 880
  kc1 = 5
  kc2 = 5
  kvdepth = 0.005
  kvrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 1
  ivfn = 1

  al fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*fmb3, fmmetal, fmpercfl, fmrhode, fmwurlie*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# fmmetal

fmmetal — Utilise la synthèse FM pour créer un son de « Heavy Metal ».

## Description

Utilise la synthèse FM pour créer un son de « Heavy Metal ». Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \  
      ifn4, ivfn
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- *twopeaks.aiff* [examples/twopeaks.aiff]
- *ifn3* -- *twopeaks.aiff* [examples/twopeaks.aiff]
- *ifn4* -- onde sinus



### Note

Le fichier « *twopeaks.aiff* » est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 3

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode *fmmetal*. Il utilise les fichiers *fmmetal.csd* [examples/fmmetal.csd] et *two-peaks.aiff* [examples/twopeaks.aiff].

### Exemple 190. Exemple de l'opcode *fmmetal*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmmetal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 10000
  kfreq = 440
  kc1 = 6
  kc2 = 5
  kvdepth = 0
  kvrate = 0
  ifn1 = 1
  ifn2 = 2
  ifn3 = 2
  ifn4 = 1
  ivfn = 1

  al fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a normal sine wave.
f 1 0 32768 10 1
; Table #2, the "twopeaks.aiff" audio file.
f 2 0 256 1 "twopeaks.aiff" 0 0 0

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

**voir Aussi**

*fmb3, fmbell, fmpercfl, fmrhode, fmwurlie*

## Crédits

Auteur : John ffitich (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# fmpercfl

fmpercfl — Utilise la synthèse FM pour créer un son de flûte percussive.

## Description

Utilise la synthèse FM pour créer un son de flûte percussive. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \  
      ifn3, ifn4, ivfn
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 4

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples



Voici un exemple de l'opcode `fmpercfl`. Il utilise le fichier `fmpercfl.csd` [examples/fmpercfl.csd].

### Exemple 191. Exemple de l'opcode `fmpercfl`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmpercfl.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 220
  kc1 = 5
  kc2 = 5
  kvdepth = 0.005
  kvrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 1
  ivfn = 1

  al fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*fmb3, fmbell, fmmetal, fmrhode, fmwurlie*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# fmrhode

fmrhode — Utilise la synthèse FM pour créer un son de piano électrique Fender Rhodes.

## Description

Utilise la synthèse FM pour créer un son de piano électrique Fender Rhodes. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \  
ifn3, ifn4, ivfn
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- *fwavblnk.aiff* [examples/fwavblnk.aiff]



### Note

Le fichier « *fwavblnk.aiff* » est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode *fmrhode*. Il utilise les fichiers *fmrhode.csd* [examples/fmrhode.csd] et *fwavblnk.aiff* [examples/fwavblnk.aiff].

### Exemple 192. Exemple de l'opcode *fmrhode*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmrhode.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 220
  kc1 = 6
  kc2 = 0
  kvdepth = 0.01
  kvrate = 3
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 2
  ivfn = 1

  al fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1
; Table #2, the "fwavblnk.aiff" audio file.
f 2 0 256 1 "fwavblnk.aiff" 0 0 0

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*fmb3, fmbell, fmmetal, fmpercfl, fmwurlie*

## Crédits

Auteur : John ffitich (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# fmvoice

fmvoice — Synthèse FM d'une Voix de Chanteur

## Description

Synthèse FM d'une Voix de Chanteur

## Syntaxe

```
ares fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, \
      ifn2, ifn3, ifn4, ivibfn
```

## Initialisation

*ifn1, ifn2, ifn3, ifn4* -- Tables, normalement des formes d'onde sinus.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kvowel* -- La voyelle chantée, dans l'intervalle 0-64

*ktilt* -- La pente spectrale du son dans l'intervalle 0 à 99

*kvibamt* -- Largeur du vibrato

*kvibrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode fmvoice. Il utilise le fichier *fmvoice.csd* [examples/fmvoice.csd].

### Exemple 193. Exemple de l'opcode fmvoice.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o fmvoice.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```

nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 110
  ; Use the fourth p-field for the vowel.
  kvowel = p4
  ktilt = 0
  kvibamt = 0.005
  kvibrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 1
  ivibfn = 1

  a1 fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, ifn2, ifn3, ifn4, ivibfn
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = vowel (a value from 0 to 64)
; Play Instrument #1 for one second, vowel=1.
i 1 0 1 1
; Play Instrument #1 for one second, vowel=2.
i 1 1 1 2
; Play Instrument #1 for one second, vowel=3.
i 1 2 1 3
; Play Instrument #1 for one second, vowel=4.
i 1 3 1 4
; Play Instrument #1 for one second, vowel=5.
i 1 4 1 5
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 University of Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# fmwurlie

fmwurlie — Utilise la synthèse FM pour créer un son de piano électrique Wurlitzer.

## Description

Utilise la synthèse FM pour créer un son de piano électrique Wurlitzer. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \  
      ifn4, ivfn
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- *fwavblnk.aiff* [examples/fwavblnk.aiff]



### Note

Le fichier « *fwavblnk.aiff* » est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5



*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode *fmwurlie*. Il utilise les fichiers *fmwurlie.csd* [examples/fmwurlie.csd] et *fwavblnk.aiff* [examples/fwavblnk.aiff].

### Exemple 194. Exemple de l'opcode *fmwurlie*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fmwurlie.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 440
  kc1 = 6
  kc2 = 1
  kvdepth = 0.005
  kvrate = 6
  ifn1 = 1
  ifn2 = 1
  ifn3 = 1
  ifn4 = 2
  ivfn = 1

  al fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 32768 10 1
; Table #2, the "fwavblnk.aiff" audio file.
f 2 0 256 1 "fwavblnk.aiff" 0 0 0

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*fmb3, fmbell, fmmetal, fmpercfl, fmrhode*

## Crédits

Auteur : John ffitich (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# fof

fof — Produit des grains FOF (sinusoïde amortie) pour la synthèse par formant et la synthèse granulaire.

## Description

La sortie audio est une succession de grains FOF (fonction d'onde formantique) amorcés à la fréquence *xfund* avec une pointe spectrale à *xform*. Pour *xfund* supérieur à 25 Hz ces grains produisent un formant comme dans la parole avec des caractéristiques spectrales déterminées par les paramètres d'entrée de taux-k. Pour des fondamentales plus basses ce générateur fournit une forme spéciale de synthèse granulaire.

## Syntaxe

```
ares fof xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \  
      ifna, ifnb, itotdur [, iphs] [, ifmode] [, iskip]
```

## Initialisation

*iolaps* -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la fréquence, et l'espace requis dépend de la valeur maximale de *xfund* \* *kdur*. La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

*ifna*, *ifnb* -- numéro de table de deux fonctions. La première est une table sinus pour la synthèse des grains FOF (une taille d'au moins 4096 est recommandée). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains FOF ; cette forme peut être linéaire (*GEN07*) ou bien sigmoïde (*GEN19*).

*itotdur* -- durée totale durant laquelle ce *fof* sera actif. Fixée normalement à p3. Aucun nouveau grain FOF n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

*iphs* (facultatif, par défaut 0) -- phase initiale du fondamental, exprimée comme une fraction d'une période (0 à 1). La valeur par défaut est 0.

*ifmode* (facultatif, par défaut 0) -- mode fréquentiel du formant. S'il est nul, chaque grain FOF garde la fréquence *xform* avec laquelle il a été amorcé. Sinon, chaque grain FOF est influencé par *xform* en continu. La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

## Exécution

*xamp* -- amplitude de crête de chaque grain FOF, observée à la toute fin de son attaque. L'attaque pourra dépasser cette valeur si l'on a une grande largeur de bande (disons  $Q < 10$ ) et/ou quand les grains FOF se superposent en partie.

*xfund* -- la fréquence fondamentale (en Hertz) des impulsions qui créent les nouveaux grains FOF.

*xform* -- la fréquence du formant, c'est-à-dire la fréquence du grain FOF induit par chaque impulsion *xfund*. Cette fréquence peut être fixe pour chaque grain ou varier en continu (voir *ifmode*).

*koct* -- indice d'octavation, normalement zéro. S'il est supérieur à zéro, il abaisse la fréquence *xfund* effective en atténuant les grains FOF de rang impair. Les nombres entiers sont des octaves, les fractions sont transitoires.

*kband* -- la largeur de bande du formant (à -6dB), exprimée en Hz. La largeur de bande détermine la vitesse de décroissance exponentielle du grain FOF, avant l'application de l'enveloppe décrite ci-dessous.

*kris*, *kdur*, *kdec* -- attaque, durée globale et chute (en secondes) du grain FOF. Ces valeurs appliquent une enveloppe à chaque grain, à la manière du générateur de Csound *linen* mais avec des formes d'attaque et de chute dessinées à partir de l'entrée *ifnb*. *kris* détermine en proportion inverse la largeur de jupe (à -40 dB) de la région formantique induite. *kdur* affecte la densité des chevauchements des grains FOF, et par conséquent la vitesse de calcul. Des valeurs typiques pour une imitation vocale sont 0,003, 0,02, 0,007.

Le générateur *fof* de Csound est inspiré du codage en C par Michael Clarke du programme *CHANT* de l'IRCAM (Xavier Rodet et al.). Chaque *fof* produit un seul formant, et les sorties de quatre ou plus de ceux-ci peuvent être additionnées pour produire une riche imitation vocale. La synthèse *fof* est une forme spéciale de la synthèse granulaire, et cette implémentation facilite la transformation entre l'imitation vocale et les textures granulaires. La vitesse de calcul dépend de *kdur*, *xfund*, et de la densité des chevauchements.

## Exemples

Voici un exemple de l'opcode *fof*. Il utilise le fichier *fof.csd* [examples/fof.csd].

### Exemple 195. Exemple de l'opcode *fof*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fof.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Combine five formants together to create
; an alto-"a" sound.

; Values common to all of the formants.
kfund init 261.659
kocf init 0
kris init 0.003
kdur init 0.02
kdec init 0.007
iolaps = 14850
ifna = 1
ifnb = 2
itotdur = p3

; First formant.
klamp = ampdb(0)
```

```
k1form init 800
k1band init 80

; Second formant.
k2amp = ampdb(-4)
k2form init 1150
k2band init 90

; Third formant.
k3amp = ampdb(-20)
k3form init 2800
k3band init 120

; Fourth formant.
k4amp = ampdb(-36)
k4form init 3500
k4band init 130

; Fifth formant.
k5amp = ampdb(-60)
k5form init 4950
k5band init 140

a1 fof k1amp, kfund, k1form, koct, k1band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur
a2 fof k2amp, kfund, k2form, koct, k2band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur
a3 fof k3amp, kfund, k3form, koct, k3band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur
a4 fof k4amp, kfund, k4form, koct, k4band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur
a5 fof k5amp, kfund, k5form, koct, k5band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur

; Combine all of the formants together.
out (a1+a2+a3+a4+a5) * 16384
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 4096 10 1
; Table #2.
f 2 0 1024 19 0.5 0.5 270 0.5

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

Les valeurs de formant pour le "a" en voix d'alto proviennent de l'*Appendice Valeurs de Formant*.

## Voir Aussi

*fof2*, *Appendice Valeurs de Formant*

## Crédits

Ajouté dans la version 1 (1990)

## fof2

**fof2** — Produit des grains FOF (sinusoïde amortie) incluant une indexation incrémentielle de taux-k avec chaque grain.

## Description

La sortie audio est une succession de grains FOF (fonction d'onde formantique) amorcés à la fréquence *xfund* avec une pointe spectrale à *xform*. Pour *xfund* supérieur à 25 Hz ces grains produisent un formant comme dans la parole avec des caractéristiques spectrales déterminées par les paramètres d'entrée de taux-k. Pour des fondamentales plus basses ce générateur fournit une forme spéciale de synthèse granulaire.

*fof2* implémente une indexation incrémentielle de taux-k dans la fonction *ifna* avec chaque grain successif.

## Syntaxe

```
ares fof2 xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
      ifna, ifnb, itotdur, kphs, kgliss [, iskip]
```

## Initialisation

*iolaps* -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la fréquence, et l'espace requis dépend de la valeur maximale de *xfund* \* *kdur*. La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

*ifna*, *ifnb* -- numéro de table de deux fonctions. La première est une table sinus pour la synthèse des grains FOF (une taille d'au moins 4096 est recommandée). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains FOF ; cette forme peut être linéaire (*GEN07*) ou bien sigmoïde (*GEN19*).

*itotdur* -- durée totale durant laquelle ce *fof* sera actif. Fixée normalement à p3. Aucun nouveau grain FOF n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

*iskip* (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

## Exécution

*xamp* -- amplitude de crête de chaque grain FOF, observée à la toute fin de son attaque. L'attaque pourra dépasser cette valeur si l'on a une grande largeur de bande (disons  $Q < 10$ ) et/ou quand les grains FOF se superposent en partie.

*xfund* -- la fréquence fondamentale (en Hertz) des impulsions qui créent les nouveaux grains FOF.

*xform* -- la fréquence du formant, c'est-à-dire la fréquence du grain FOF induit par chaque impulsion *xfund*. Cette fréquence peut être fixe pour chaque grain ou varier en continu (voir *ifmode*).

*koct* -- indice d'octaviation, normalement zéro. S'il est supérieur à zéro, il abaisse la fréquence *xfund* effective en atténuant les grains FOF de rang impair. Les nombres entiers sont des octaves, les fractions sont transitoires.

*kband* -- la largeur de bande du formant (à -6dB), exprimée en Hz. La largeur de bande détermine la vitesse de décroissance exponentielle du grain FOF, avant l'application de l'enveloppe décrite ci-dessous.

*kris*, *kdur*, *kdec* -- attaque, durée globale et chute (en secondes) du grain FOF. Ces valeurs appliquent une enveloppe à chaque grain, à la manière du générateur de Csound *linen* mais avec des formes d'attaque et de chute dessinées à partir de l'entrée *ifnb*. *kris* détermine en proportion inverse la largeur de jupe (à -40 dB) de la région formantique induite. *kdur* affecte la densité des chevauchements des grains FOF, et par conséquent la vitesse de calcul. Des valeurs typiques pour une imitation vocale sont 0,003, 0,02, 0,007.

*kphs* -- permet d'indexer au taux-k la table de fonction *ifna* avec chaque grain successif, ce qui permet d'appliquer le recalage temporel. Les valeurs de *kphs* sont normalisées entre 0 et 1, 1 étant la fin de la table de fonction *ifna*.

*kgliss* -- fixe la hauteur finale de chaque grain en fonction de sa hauteur initiale, en octaves. Ainsi *kgliss* = 2 signifie que le grain se termine deux octaves plus haut que sa hauteur initiale, tandis qu'avec *kgliss* = -3/4 le grain se termine une sixte majeure plus bas. Chaque 1/12 ajouté à *kgliss* élève la hauteur finale d'un demi-ton. Si vous ne voulez pas de glissando, fixez *kgliss* à 0.

Le générateur *fof* de Csound est inspiré du codage en C par Michael Clarke du programme *CHANT* de l'IRCAM (Xavier Rodet et al.). Chaque *fof* produit un seul formant, et les sorties de quatre ou plus de ceux-ci peuvent être additionnées pour produire une riche imitation vocale. La synthèse *fof* est une forme spéciale de la synthèse granulaire, et cette implémentation facilite la transformation entre l'imitation vocale et les textures granulaires. La vitesse de calcul dépend de *kdur*, *xfund*, et de la densité des chevauchements.



## Note

La fréquence finale de chaque grain étant égale à  $kform * (2 ^ kgliss)$ , des valeurs trop importantes de *kgliss* pourront provoquer un repliement. Par exemple, *kform* = 3000 et *kgliss* = 3 placent la fréquence finale au-delà de la fréquence de Nyquist si *sr* = 44100. Il est prudent de pondérer *kgliss* en conséquence.

## Exemples

Voici un exemple de l'opcode *fof2*. Il utilise le fichier *fof2.csd* [examples/fof2.csd].

### Exemple 196. Exemple de l'opcode *fof2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o fof2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

;By Andres Cabrera 2007

instr 1          ;table-lookup vocal synthesis
```

```

kris init p12
kdur init p13
kdec init p14

iolaps init p15

ifna init 1 ; Sine wave
ifnb init 2 ; Straight line rise shape

itotdur init p3

kphs init 0 ; No phase modulation (constant kphs)

kfund line p4, p3, p5
kform line p6, p3, p7
koc1 line p8, p3, p9
kband line p10, p3, p11
kg1iss line p16, p3, p17

kenv linen 5000, 0.03, p3, 0.03 ;to avoid clicking

aout fof2 kenv, kfund, kform, koc1, kband, kris, kdur, kdec, iolaps, \
      ifna, ifnb, itotdur, kphs, kg1iss

outs aout, aout
endin

</CsInstruments>
<CsScore>
f1 0 8192 10 1
f2 0 4096 7 0 4096 1

;          kfund1 kfund2 kform1 kform2 koc1 koc2 kband1 kband2 kris kdur kdec iolaps kg
i1 0 4 220 220 510 510 0 0 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 910 0 0 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 0 100 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 1 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 0 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 0 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 0 30 30 0.01 0.05 0.01 100
i1 + . 220 440 510 510 0 0 30 30 0.01 0.05 0.01 100

e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode fof2, qui produit des sons de voyelle en utilisant des formants générés par fof2 avec les valeurs de l'appendice *Valeurs de Formant*. Il utilise le fichier *fof2-2.csd* [examples/fof2-2.csd].

### Exemple 197. Exemple de l'opcode fof2 pour produire des sons de voyelle.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in
-odac -iadc ;;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o fof2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

; Example by Chuckk Hubbard 2007

```



```

instr 1                                ;table-lookup vocal synthesis

iolaps = 120
ifna = 1                               ;f1 - sine wave
ifnb = 2                               ;f2 - linear rise shape
itotdur = p3
iamp = p4 * 0dbfs
ifreq1 = p5                             ;starting frequency
ifreq2 = p6                             ;ending frequency

kamp linseg 0, .003, iamp, itotdur-.007, iamp, .003, 0, .001, 0
kfund expseg ifreq1, itotdur, ifreq2
koct init 0
kris init .003
kdur init .02
kdec init .007
kphs init 0
kgliiss init 0

iforma = p7                             ;starting spectrum
iformb = p8                             ;ending spectrum

iform1a tab_i 0, iforma                 ;read values of 5 formants for 1st spectrum
iform2a tab_i 1, iforma
iform3a tab_i 2, iforma
iform4a tab_i 3, iforma
iform5a tab_i 4, iforma
idb1a tab_i 5, iforma                  ;read decibel levels for same 5 formants
idb2a tab_i 6, iforma
idb3a tab_i 7, iforma
idb4a tab_i 8, iforma
idb5a tab_i 9, iforma
iband1a tab_i 10, iforma               ;read bandwidths for same 5 formants
iband2a tab_i 11, iforma
iband3a tab_i 12, iforma
iband4a tab_i 13, iforma
iband5a tab_i 14, iforma
iamp1a = ampdb(idb1a)                  ;convert db to linear multipliers
iamp2a = ampdb(idb2a)
iamp3a = ampdb(idb3a)
iamp4a = ampdb(idb4a)
iamp5a = ampdb(idb5a)

iform1b tab_i 0, iformb                ;values of 5 formants for 2nd spectrum
iform2b tab_i 1, iformb
iform3b tab_i 2, iformb
iform4b tab_i 3, iformb
iform5b tab_i 4, iformb
idb1b tab_i 5, iformb                  ;decibel levels for 2nd set of formants
idb2b tab_i 6, iformb
idb3b tab_i 7, iformb
idb4b tab_i 8, iformb
idb5b tab_i 9, iformb
iband1b tab_i 10, iformb               ;bandwidths for 2nd set of formants
iband2b tab_i 11, iformb
iband3b tab_i 12, iformb
iband4b tab_i 13, iformb
iband5b tab_i 14, iformb
iamp1b = ampdb(idb1b)                  ;convert db to linear multipliers
iamp2b = ampdb(idb2b)
iamp3b = ampdb(idb3b)
iamp4b = ampdb(idb4b)
iamp5b = ampdb(idb5b)

kform1 line iform1a, itotdur, iform1b   ;transition between formants
kform2 line iform2a, itotdur, iform2b
kform3 line iform3a, itotdur, iform3b
kform4 line iform4a, itotdur, iform4b
kform5 line iform5a, itotdur, iform5b
kband1 line iband1a, itotdur, iband1b   ;transition of bandwidths
kband2 line iband2a, itotdur, iband2b
kband3 line iband3a, itotdur, iband3b
kband4 line iband4a, itotdur, iband4b
kband5 line iband5a, itotdur, iband5b
kamp1 line iamp1a, itotdur, iamp1b       ;transition of amplitudes of formants
kamp2 line iamp2a, itotdur, iamp2b
kamp3 line iamp3a, itotdur, iamp3b
kamp4 line iamp4a, itotdur, iamp4b
kamp5 line iamp5a, itotdur, iamp5b

;5 formants for each spectrum

```

```

a1    fof2    kamp1, kfund, kform1, koct, kband1, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a2    fof2    kamp2, kfund, kform2, koct, kband2, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a3    fof2    kamp3, kfund, kform3, koct, kband3, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a4    fof2    kamp4, kfund, kform4, koct, kband4, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a5    fof2    kamp5, kfund, kform5, koct, kband5, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,

aout   =      (a1+a2+a3+a4+a5) * kamp/5      ;sum and scale

aenv    linen 1, 0.05, p3, 0.05      ;to avoid clicking

      outs      aout*aenv, aout*aenv
      endin

</CsInstruments>
<CsScore>
f1 0 8192 10 1
f2 0 4096 7 0 4096 1

;*****
; tables of formant values adapted from MiscFormants.html
; 100's: soprano    200's: alto    300's: countertenor    400's: tenor    500's: bass
; -01: "a" sound    -02: "e" sound    -03: "i" sound    -04: "o" sound    -05: "u" sound
; p-5 through p-9: frequencies of 5 formants
; p-10 through p-14: decibel levels of 5 formants
; p-15 through p-19: bandwidths of 5 formants

;          formant frequencies          decibel levels          bandwidths
;soprano
f101 0 16 -2    800    1150    2900    3900    4950    0.001    -6    -32    -20    -50
f102 0 16 -2    350    2000    2800    3600    4950    0.001    -20    -15    -40    -56
f103 0 16 -2    270    2140    2950    3900    4950    0.001    -12    -26    -26    -44
f104 0 16 -2    450    800    2830    3800    4950    0.001    -11    -22    -22    -50
f105 0 16 -2    325    700    2700    3800    4950    0.001    -16    -35    -40    -60
;alto
f201 0 16 -2    800    1150    2800    3500    4950    0.001    -4    -20    -36    -60
f202 0 16 -2    400    1600    2700    3300    4950    0.001    -24    -30    -35    -60
f203 0 16 -2    350    1700    2700    3700    4950    0.001    -20    -30    -36    -60
f204 0 16 -2    450    800    2830    3500    4950    0.001    -9    -16    -28    -55
f205 0 16 -2    325    700    2530    3500    4950    0.001    -12    -30    -40    -64
;countertenor
f301 0 16 -2    660    1120    2750    3000    3350    0.001    -6    -23    -24    -38
f302 0 16 -2    440    1800    2700    3000    3300    0.001    -14    -18    -20    -20
f303 0 16 -2    270    1850    2900    3350    3590    0.001    -24    -24    -36    -36
f304 0 16 -2    430    820    2700    3000    3300    0.001    -10    -26    -22    -34
f305 0 16 -2    370    630    2750    3000    3400    0.001    -20    -23    -30    -34
;tenor
f401 0 16 -2    650    1080    2650    2900    3250    0.001    -6    -7    -8    -22
f402 0 16 -2    400    1700    2600    3200    3580    0.001    -14    -12    -14    -20
f403 0 16 -2    290    1870    2800    3250    3540    0.001    -15    -18    -20    -30
f404 0 16 -2    400    800    2600    2800    3000    0.001    -10    -12    -12    -26
f405 0 16 -2    350    600    2700    2900    3300    0.001    -20    -17    -14    -26
;bass
f501 0 16 -2    600    1040    2250    2450    2750    0.001    -7    -9    -9    -20
f502 0 16 -2    400    1620    2400    2800    3100    0.001    -12    -9    -12    -18
f503 0 16 -2    250    1750    2600    3050    3340    0.001    -30    -16    -22    -28
f504 0 16 -2    400    750    2400    2600    2900    0.001    -11    -21    -20    -40
f505 0 16 -2    350    600    2400    2675    2950    0.001    -20    -32    -28    -36
;*****

;          start dur amp          start freq          end freq          start formant          end formant
i1    0 1 .8    440    412.5    201    203
i1    + . .8    412.5    550    201    204
i1    + . .8    495    330    202    205

i1    + . .8    110    103.125    501    503
i1    + . .8    103.125 137.5    501    504
i1    + . .8    123.75 82.5    502    505

i1    7 . .4    440    412.5    201    203
i1    8 . .4    412.5    550    201    204
i1    9 . .4    495    330    202    205
i1    7 . .4    110    103.125    501    503
i1    8 . .4    103.125 137.5    501    504
i1    9 . .4    123.75 82.5    502    505
i1    + . .4    440    412.5    101    103
i1    + . .4    412.5    550    101    104
i1    + . .4    495    330    102    105
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*fof*

## Crédits

Auteur : Rasmus Ekman

*fof2* est une modification de *fof* par Rasmus Ekman

Nouveau dans Csound 3.45

# fofilter

fofilter — Formant filter.

## Description

Fofilter generates a stream of overlapping sinewave grains, when fed with a pulse train. Each grain is the impulse response of a combination of two BP filters. The grains are defined by their attack time (determining the skirtwidth of the formant region at -60dB) and decay time (-6dB bandwidth). Overlapping will occur when  $1/\text{freq} < \text{decay}$ , but, unlike FOF, there is no upper limit on the number of overlaps. The original idea for this opcode came from J McCartney's formlet class in SuperCollider, but this is possibly implemented differently(?).

## Syntax

```
asig fofilter ain, kcf, kris, kdec[, istor]
```

## Initialization

*istor* --initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*asig* -- input signal.

*kcf* -- filter centre frequency

*kris* -- impulse response attack time (secs).

*kdec* -- impulse response decay time (secs).

## Examples

### Exemple 198. Example

```
kfe      expseg 10, p3*0.9, 180, p3*0.1, 175
kenv     linen 1000, 0.05, p3, 0.05
asig     buzz kenv, kfe, sr/(2*kfe), 1
afil     fofilter asig, 900, 0.007, 0.04

out afil
```

## Credits

Author: Victor Lazzarini  
January 2005

New plugin in version 5

January 2005.

# fog

fog — La sortie audio est une succession de grains obtenus à partir des données d'une table de fonction.

## Description

La sortie audio est une succession de grains obtenus à partir des données de la table de fonction *ifna*. L'enveloppe locale de ces grains et leur distribution temporelle sont basées sur le modèle de la synthèse *fof* et permettent un contrôle détaillé de la synthèse granulaire.

## Syntaxe

```
ares fog xamp, xdens, xtrans, aspd, koct, kband, kris, kdur, kdec, \  
      iolaps, ifna, ifnb, itotdur [, iphs] [, itmode] [, iskip]
```

## Initialisation

*iolaps* -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la densité, et l'espace requis dépend de la valeur maximale de  $xdens * kdur$ . La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

*ifna*, *ifnb* -- numéros de table de deux fonctions. La première contient les données utilisées pour la granulation, provenant habituellement d'un fichier son (*GEN01*). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains ; cette forme est normalement une sigmoïde (*GEN19*) mais elle peut être aussi linéaire (*GEN05*).

*itotdur* -- durée totale durant laquelle ce *fog* sera actif. Fixée normalement à p3. Aucun nouveau grain n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

*iphs* (facultatif) -- phase initiale du fondamental, exprimée comme une fraction d'une période (0 à 1). La valeur par défaut est 0.

*itmode* (facultatif) -- type de transposition. S'il est nul, chaque grain garde la valeur *xtrans* avec laquelle il a été amorcé. Sinon, chaque grain est influencé par *xtrans* de manière continue. La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

## Exécution

*xamp* -- facteur d'amplitude. L'amplitude dépend également du nombre de grains se chevauchant, de l'interaction de la forme montante (*ifnb*) et de la chute exponentielle (*kband*), et des valeurs de la forme d'onde du grain (*ifna*). L'amplitude réelle peut ainsi dépasser *xamp*.

*xdens* -- densité. Nombre de grains par seconde.

*xtrans* -- facteur de transposition. Le taux de lecture des données de la table de fonction *ifna* dans chaque grain. Il a pour effet de transposer le matériel original. Une valeur de 1 produit la hauteur originale. Les valeurs supérieures à 1 transposent vers le haut tandis que les valeurs inférieures à 1 le font vers le bas. Les valeurs négatives provoquent une lecture à l'envers de la table.

*aspd* -- indice de lecture initial. *aspd* est l'indice de lecture normalisé (0 à 1) dans la table *ifna* qui détermine le mouvement d'un pointeur à partir duquel commence la lecture dans chaque grain. (*xtrans* détermine le taux de lecture des données à partir de ce pointeur.)

*koct* -- indice d'octaviation. Ce paramètre fonctionne de manière identique à celui qui est décrit dans *fof*.

*kband*, *kris*, *kdur*, *kdec* -- forme de l'enveloppe du grain. Ces paramètres déterminent les temps de décroissance exponentielle (*kband*), et d'attaque (*kris*), la durée totale (*kdur*), et celle de la chute (*kdec*) de l'enveloppe du grain. Leur mode opératoire est identique à celui des paramètres d'enveloppe locale dans *fof*.

## Exemples

```
;p4 = facteur de transposition
;p5 = facteur de vitesse
;p6 = table de fonction pour les données du grain
il = sr/ftlen(p6) ; prise en compte du taux d'échantillonnage et de la longueur de la table
a1 phasor il*p5 ; indice pour la vitesse
a2 fog 5000, 100, p4, a1, 0, 0, , .01, .02, .01, 2, p6, 1, p3, 0, 1
```

## Crédits

Auteur : Michael Clark  
Huddersfield  
Mai 1997

Nouveau dans la version 3.46

Le générateur *fog* de Csound a été écrit par Michael Clarke, comme suite à ses travaux antérieurs basés sur l'algorithme FOF de l'IRCAM.

Notes ajoutées par Rasmus Ekman en septembre 2002.

# fold

fold — Adds artificial foldover to an audio signal.

## Description

Adds artificial foldover to an audio signal.

## Syntax

```
ares fold asig, kincr
```

## Performance

*asig* -- input signal

*kincr* -- amount of foldover expressed in multiple of sampling rate. Must be  $\geq 1$

*fold* is an opcode which creates artificial foldover. For example, when *kincr* is equal to 1 with *sr*=44100, no foldover is added. When *kincr* is set to 2, the foldover is equivalent to a downsampling to 22050, when it is set to 4, to 11025 etc. Fractional values of *kincr* are possible, allowing a continuous variation of foldover amount. This can be used for a wide range of special effects.

## Examples

Here is an example of the fold opcode. It uses the file *fold.csd* [examples/fold.csd].

### Exemple 199. Example of the fold opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o fold.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use an ordinary sine wave.
asig oscils 30000, 100, 1

; Vary the fold-over amount from 1 to 200.
kincr line 1, p3, 200
a1 fold asig, kincr

out a1
```



```
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for four seconds.
i 1 0 4

e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Gabriel Maldonado  
Italy  
1999

New in Csound version 3.56

# follow

follow — Envelope follower unit generator.

## Description

Envelope follower unit generator.

## Syntax

```
ares follow asig, idt
```

## Initialization

*idt* -- This is the period, in seconds, that the average amplitude of *asig* is reported. If the frequency of *asig* is low then *idt* must be large (more than half the period of *asig* )

## Performance

*asig* -- This is the signal from which to extract the envelope.

## Examples

Here is an example of the follow opcode. It uses the file *follow.csd* [examples/follow.csd], and *beats.wav* [examples/beats.wav].

### Exemple 200. Example of the follow opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o follow.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - play a WAV file.
instr 1
  al soundin "beats.wav"
  out al
endin

; Instrument #2 - have another waveform follow the WAV file.
instr 2
  ; Follow the WAV file.
  as soundin "beats.wav"
```

```
af follow as, 0.01

; Use a sine waveform.
as oscil 4000, 440, 1
; Have it use the amplitude of the followed WAV file.
al balance as, af

out al
endin

</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

To avoid zipper noise, by discontinuities produced from complex envelope tracking, a lowpass filter could be used, to smooth the estimated envelope.

## Credits

Author: Paris Smaragdis  
MIT, Cambridge  
1995

# follow2

follow2 — Another controllable envelope extractor.

## Description

A controllable envelope extractor using the algorithm attributed to Jean-Marc Jot.

## Syntax

```
ares follow2 asig, katt, krel
```

## Performance

*asig* -- the input signal whose envelope is followed

*katt* -- the attack rate (60dB attack time in seconds)

*krel* -- the decay rate (60dB decay time in seconds)

The output tracks the amplitude envelope of the input signal. The rate at which the output grows to follow the signal is controlled by the *katt*, and the rate at which it decreases in response to a lower amplitude, is controlled by the *krel*. This gives a smoother envelope than *follow*.

## Examples

Here is an example of the follow2 opcode. It uses the file *follow2.csd* [examples/follow2.csd], and *beats.wav* [examples/beats.wav].

### Exemple 201. Example of the follow2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o follow2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - play a WAV file.
instr 1
  a1 soundin "beats.wav"
  out a1
endin

; Instrument #2 - have another waveform follow the WAV file.
```

```
instr 2
; Follow the WAV file.
as soundin "beats.wav"
af follow2 as, 0.01, 0.1

; Use a noise waveform.
ar rand 44100
; Have it use the amplitude of the followed WAV file.
al balance ar, af

out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: John ffitch  
The algorithm for the *follow2* is attributed to Jean-Marc Jot.  
University of Bath, Codemist Ltd.  
Bath, UK  
February 2000

Example written by Kevin Conder.

New in Csound version 4.03

Added notes by Rasmus Ekman on September 2002.

# foscil

foscil — Un oscillateur élémentaire modulé en fréquence.

## Description

Un oscillateur élémentaire modulé en fréquence.

## Syntaxe

```
ares foscil xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction. Nécessite un point de garde de lecture cyclique.

*iphs* (facultatif, par défaut 0) -- phase initiale de la forme d'onde dans la table *ifn*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- un dénominateur commun, en cycles par seconde, pour les fréquences porteuse et modulante.

*xcar* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la porteuse.

*xmod* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la modulante.

*kndx* -- l'indice de modulation.

*foscil* est une unité composée qui assemble deux opcodes *oscil* dans la configuration familière de synthèse FM de Chowning, où la sortie au taux audio de l'un des générateurs est utilisée pour moduler l'entrée en fréquence de l'autre (la « porteuse »). Fréquence de la porteuse =  $kcps * xcar$  et fréquence modulante =  $kcps * xmod$ . Pour des valeurs entières de *xcar* et de *xmod*, la fondamentale perçue sera la valeur positive minimale de  $kcps * (xcar - n * xmod)$ ,  $n = 0, 1, 2, \dots$  L'entrée *kndx* est l'indice de modulation (habituellement variant dans le temps approximativement dans l'intervalle de 0 à 4) qui détermine la distribution de l'énergie acoustique parmi les positions des partiels données par  $n = 0, 1, 2, \dots$ , etc. *ifn* doit pointer sur une onde sinus stockée. Avant la version 3.50, *xcar* et *xmod* ne pouvaient être que de taux-k.

La formule utilisée pour cette implémentation de la synthèse FM est  $xamp * \cos(2\pi * t * kcps * xcar + kndx * \sin(2\pi * t * kcps * xmod) - \#)$ , en supposant que la table est une onde sinus.

## Exemples

Voici un exemple de l'opcode foscil. Il utilise le fichier *foscil.csd* [exemples/foscil.csd].

### Exemple 202. Exemple de l'opcode foscil.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o foscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic FM waveform.
instr 1
  kamp = 10000
  kcps = 440
  kcar = 600
  kmod = 210
  kndx = 2
  ifn = 1

  al foscil kamp, kcps, kcar, kmod, kndx, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Exemple écrit par Kevin Conder.

# foscili

foscili — Oscillateur élémentaire modulé en fréquence avec interpolation linéaire.

## Description

Oscillateur élémentaire modulé en fréquence avec interpolation linéaire.

## Syntaxe

```
ares foscili xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction. Nécessite un point de garde de lecture cyclique.

*iphs* (facultatif, par défaut 0) -- phase initiale de la forme d'onde dans la table *ifn*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- un dénominateur commun, en cycles par seconde, pour les fréquences porteuse et modulante.

*xcar* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la porteuse.

*xmod* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la modulante.

*kndx* -- l'indice de modulation.

*foscili* diffère de *foscil* en ce que la procédure standard d'utilisation d'une phase tronquée comme index de lecture des échantillons est remplacée ici par une interpolation entre deux lectures successives. Les générateurs avec interpolation produiront un signal de sortie nettement plus propre, mais ils peuvent prendre jusqu'à deux fois plus de temps de calcul. On peut obtenir également ce type de précision sans le surcoût du calcul de l'interpolation en utilisant de grandes tables de fonction stockées de 2K, 4K ou 8K points, si l'on dispose de cet espace mémoire.

## Exemples

Voici un exemple de l'opcode *foscili*. Il utilise le fichier *foscili.csd* [examples/foscili.csd].

### Exemple 203. Exemple de l'opcode *foscili*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```



```

; Audio out   Audio in   No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o foscili.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic FM waveform.
instr 1
  kamp = 10000
  kcps = 440
  kcar = 600
  kmod = 210
  kndx = 2
  ifn = 1

  al foscil kamp, kcps, kcar, kmod, kndx, ifn
  out al
endin

; Instrument #2 - the basic FM waveform with extra interpolation.
instr 2
  kamp = 10000
  kcps = 440
  kcar = 600
  kmod = 210
  kndx = 2
  ifn = 1

  al foscili kamp, kcps, kcar, kmod, kndx, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave table with a small amount of data.
f 1 0 4096 10 1

; Play Instrument #1, the basic FM instrument, for
; two seconds. This should sound relatively rough.
i 1 0 2

; Play Instrument #2, the interpolated FM instrument, for
; two seconds. This should sound relatively smooth.
i 2 2 2
e

</CsScore>
</CsSoundSynthesizer>

```

## Crédits

Exemple écrit par Kevin Conder.

# fout

fout — Envoie des signaux de taux-a vers un nombre arbitraire de canaux dans un fichier externe.

## Description

*fout* envoie *N* signaux de taux-a vers un fichier spécifié à *N* canaux.

## Syntaxe

```
fout ifilename, iformat, aout1 [, aout2, aout3,...,aoutN]
```

## Initialisation

*ifilename* -- le nom du fichier de sortie (entre guillemets)

*iformat* -- un indicateur pour choisir le format du fichier de sortie (note : il se peut que les versions de Csound antérieures à la 5.0 ne supportent que les formats 0, 1 et 2) :

- 0 - échantillons en flottants sur 32 bit sans en-tête (fichier PCM binaire multicanaux)
- 1 - entiers sur 16 bit sans en-tête (fichier PCM binaire multicanaux)
- 2 - entiers sur 16 bit avec en-tête. Le type de l'en-tête dépend du format de restitution (-o). Par exemple, si l'utilisateur choisit le format AIFF (en utilisant l'*option -A*), le format de l'en-tête sera de type AIFF.
- 3 - échantillons u-law avec un en-tête (voir iformat=2).
- 4 - entiers sur 16 bit avec un en-tête (voir iformat=2).
- 5 - entiers sur 32 bit avec un en-tête (voir iformat=2).
- 6 - flottants sur 32 bit avec un en-tête (voir iformat=2).
- 7 - entiers non signés sur 8 bit avec un en-tête (voir iformat=2).
- 8 - entiers sur 24 bit avec un en-tête (voir iformat=2).
- 9 - flottants sur 64 bit avec un en-tête (voir iformat=2).

De plus, les versions de Csound à partir de la 5.0 permettent de choisir explicitement un type d'en-tête particulier en spécifiant le format comme 10 \* typeFichier + formatEchantillon, où typeFichier peut valoir 1 pour WAV, 2 pour AIFF, 3 pour fichiers brut (sans en-tête) et 4 pour IRCAM ; formatEchantillon est l'une des valeurs ci-dessus comprise entre 0 et 9, sauf que le format d'échantillon 0 est déduit de la ligne de commande (-o), le format 1 représente des entiers signés sur 8 bit et le format 2 est a-law. Ainsi, par exemple, *iformat = 25* signifie des entiers sur 32 bit avec un en-tête AIFF.

## Exécution

*aout1,... aoutN* -- signaux à écrire dans le fichier. Dans le cas de fichiers bruts, l'étendue de l'amplitude des signaux audio est déterminée par le format d'échantillon choisi ; pour les fichiers son avec un en-tête

comme WAV et AIFF, les signaux audio doivent être dans l'intervalle compris entre -0dbfs et 0dbfs.

*fout* (file output) écrit des échantillons de signaux audio dans un fichier avec n'importe quel nombre de canaux. Le nombre de canaux dépend du nombre de variables *aoutN* (par exemple un signal mono avec un seul argument de taux-a, un signal stéréo avec deux arguments de taux-a, etc.) Le nombre maximum de canaux est fixé à 64. Plusieurs opcodes *fout* peuvent se trouver dans le même instrument, se référant à différents fichiers.

Noter que, contrairement à *out*, *outs* et *outq*, *fout* ne remet pas à zéro la variable audio, c'est pourquoi l'on doit la remettre à zéro après l'appel. Si l'on travaille en polyphonie, on peut utiliser les opcodes *vincr* et *clear* pour cela.

Noter que *fout* et *foutk* peuvent utiliser soit une chaîne de caractères contenant un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.



## Note

Si l'on utilise *fout* pour générer un fichier audio depuis la sortie globale de Csound, il peut être désirable d'utiliser l'opcode *monitor* qui peut capter le tampon de sortie, ce qui évite le routage.

## Exemples

Voici un exemple de l'opcode *fout*. Il utilise le fichier *fout.csd* [exemples/fout.csd].

### Exemple 204. Exemple de l'opcode *fout*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fout.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  ; Create an audio signal.
  asig oscils iamp, icps, iphs

  ; Write the audio signal to a headerless audio file
  ; called "fout.raw".
  fout "fout.raw", 1, asig
endin

</CsInstruments>
<CsScore>
```

```
; Play Instrument #1 for 2 seconds.
i 1 0 2
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode `fout` avec une partition polyphonique. Il utilise les fichiers `fout_poly.csd` [examples/fout\_poly.csd] et `beats.wav` [examples/beats.wav].

### Exemple 205. Exemple de l'opcode `fout` avec une partition polyphonique.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o fout_poly.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - Play an audio file.
instr 1
; Generate an audio signal using
; the audio file "beats.wav".
asig soundin "beats.wav"

out asig
endin

; Instrument #2 - Create a basic tone.
instr 2
iamp = 5000
icps = 440
iphs = 0

; Create an audio signal.
asig oscils iamp, icps, iphs

out asig
endin

; Instrument #99 - Save the global signal to a file.
instr 99
; Read the csound output buffer
aoutput monitor
; Write the output of csound to a headerless
; audio file called "fout_poly.raw".
fout "fout_poly.raw", 1, aoutput

endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2

; Play Instrument #2 every quarter-second.
i 2 0.00 0.1
i 2 0.25 0.1
i 2 0.50 0.1
i 2 0.75 0.1
i 2 1.00 0.1
```

```
i 2 1.25 0.1
i 2 1.50 0.1
i 2 1.75 0.1

; Make sure the global instrument, #99, is running
; during the entire performance (2 seconds).
i 99 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de *fout*, qui l'utilise pour sauvegarder le contenu d'une table dans un fichier audio. Il utilise les fichiers *fout\_ftable.csd* [examples/fout\_ftable.csd] et *beats.wav* [examples/beats.wav].

### Exemple 206. Exemple de l'opcode *fout* pour sauvegarder le contenu d'une f-table.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o fout_ftable.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; By: Jonathan Murphy 2007

gilen          =          131072
gicps          =          sr/gilen
gitab          = ftgen      1, 0, gilen, 10, 1

instr 1

/***** write file to table *****/

ain            diskint2      "beats.wav", 1, 0, 1
aphs           phasor        gicps
andx           =             apha * gilen
               =             ain, andx, gitab

/***** write table to file *****/

aosc           table         apha, gitab, 1
out            aosc
fout           "beats_copy.wav", 6, aosc

endin

</CsInstruments>
<CsScore>
il 0 2
e
</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*fiopen, fouti, foutir, foutk, monitor*

## Crédits

Auteur : Gabriel Maldonado

Italie  
1999

L'exemple simple a été écrit par Kevin Conder.

Nouveau dans la version 3.56 de Csound

Octobre 2002. Note ajoutée par Richard Dobson.

# fouti

fouti — Envoie des signaux de taux-i d'un nombre arbitraire de canaux dans un fichier spécifié.

## Description

*fouti* envoie *N* signaux de taux-i vers un fichier spécifié à *N* canaux.

## Syntaxe

```
fouti ihandle, iformat, iflag, iout1 [, iout2, iout3, ..., ioutN]
```

## Initialisation

*ihandle* -- un nombre qui spécifie ce fichier.

*iformat* -- un indicateur pour choisir le format du fichier de sortie :

- 0 - flottants en format texte
- 1 - flottants sur 32 bit en format binaire

*iflag* -- choisit le mode d'écriture dans le fichier ASCII (n'est valide qu'en mode ASCII ; en mode binaire *iflag* n'a aucune signification, mais il doit quand même être présent). *iflag* peut prendre une des valeurs suivantes :

- 0 - ligne de texte sans préfixe d'instrument
- 1 - ligne de texte avec préfixe d'instrument (voir ci-dessous)
- 2 - remet à zéro le temps des préfixes d'instrument (à n'utiliser que dans certains cas particuliers. Voir ci-dessous)

*iout*, ..., *ioutN* -- valeurs à écrire dans le fichier

## Exécution

*fouti* et *foutir* écrivent des valeurs de taux-i dans un fichier. On utilise ces opcodes principalement pour générer un fichier de partition pendant une session en temps-réel. Pour cela, il faut fixer *iformat* à 0 (sortie dans un fichier texte) et *iflag* à 1, ce qui active la sortie d'un préfixe constitué des chaînes de caractères *inum*, *actiontime* et *duration*, avant les valeurs des arguments *iout1*...*ioutN*. Les arguments dans le préfixe font référence au numéro de l'instrument, à la date et à la durée de la note courante.

Noter que *fout* et *foutk* peuvent utiliser soit une chaîne de caractères contenant un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.

## Voir Aussi

*fiopen, fout, foutir, foutk*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound



# foutir

foutir — Envoie des signaux de taux-i d'un nombre arbitraire de canaux dans un fichier spécifié.

## Description

*foutir* envoie *N* signaux de taux-i vers un fichier spécifié à *N* canaux.

## Syntaxe

```
foutir ihandle, iformat, iflag, iout1 [, iout2, iout3, ..., ioutN]
```

## Initialisation

*ihandle* -- un nombre qui spécifie ce fichier.

*iformat* -- un indicateur pour choisir le format du fichier de sortie :

- 0 - flottants en format texte
- 1 - flottants sur 32 bit en format binaire

*iflag* -- choisit le mode d'écriture dans le fichier ASCII (n'est valide qu'en mode ASCII ; en mode binaire *iflag* n'a aucune signification, mais il doit quand même être présent). *iflag* peut prendre une des valeurs suivantes :

- 0 - ligne de texte sans préfixe d'instrument
- 1 - ligne de texte avec préfixe d'instrument (voir ci-dessous)
- 2 - remet à zéro le temps des préfixes d'instrument (à n'utiliser que dans certains cas particuliers. Voir ci-dessous)

*iout*, ..., *ioutN* -- valeurs à écrire dans le fichier

## Exécution

*fouti* et *foutir* écrivent des valeurs de taux-i dans un fichier. On utilise ces opcodes principalement pour générer un fichier de partition pendant une session en temps-réel. Pour cela, il faut fixer *iformat* à 0 (sortie dans un fichier texte) et *iflag* à 1, ce qui active la sortie d'un préfixe constitué des chaînes de caractères *inum*, *actiontime* et *duration*, avant les valeurs des arguments *iout1*...*ioutN*. Les arguments dans le préfixe font référence au numéro de l'instrument, à la date et à la durée de la note courante.

La différence entre *fouti* et *foutir* est que dans le cas de *fouti*, quand *iflag* vaut 1, la durée du premier op-code est indéfinie (elle est ainsi remplacée par un point). Tandis que *foutir* n'est défini qu'à la fin de la note, si bien que la ligne de texte correspondante n'est écrite qu'à la fin de la note courante (afin de connaître sa durée). Le fichier correspondant est lié par la valeur de *ihandle* générée par l'opcode *fiopen*. On peut ainsi utiliser *fouti* et *foutir* pour générer une partition Csound tout en jouant une session en temps-réel.

Noter que *fout* et *foutk* peuvent utiliser soit une chaîne de caractères contenant un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.

## Voir Aussi

*fiopen, fout, fouti, foutk*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# foutk

**foutk** — Envoie des signaux de taux-k vers un nombre arbitraire de canaux dans un fichier externe, en format brut (sans en-tête).

## Description

*foutk* envoie *N* signaux de taux-k vers un fichier spécifié à *N* canaux.

## Syntaxe

```
foutk ifilename, iformat, kout1 [, kout2, kout3, ..., koutN]
```

## Initialisation

*ifilename* -- le nom du fichier de sortie (entre guillemets)

*iformat* -- un indicateur pour choisir le format du fichier de sortie (note : il se peut que les versions de Csound antérieures à la 5.0 ne supportent que les formats 0 et 1) :

- 0 - échantillons en flottants sur 32 bit sans en-tête (fichier PCM binaire multicanaux)
- 1 - entiers sur 16 bit sans en-tête (fichier PCM binaire multicanaux)
- 2 - entiers sur 16 bit sans en-tête (fichier PCM binaire multicanaux)
- 3 - échantillons u-law sans en-tête
- 4 - entiers sur 16 bit sans en-tête
- 5 - entiers sur 32 bit sans en-tête
- 6 - flottants sur 32 bit sans en-tête
- 7 - entiers non signés sur 8 bit sans en-tête
- 8 - entiers sur 24 bit sans en-tête
- 9 - flottants sur 64 bit sans en-tête

## Exécution

*kout1, ..., koutN* -- signaux au taux de contrôle à écrire dans le fichier. L'étendue de l'amplitude des signaux est déterminée par le format d'échantillon choisi.

*foutk* opère de la même manière que *fout*, mais avec des signaux de taux-k. *iformat* peut prendre une valeur entre 0 et 9, ou 0 et 1 avec une ancienne version de Csound.

Noter que *fout* et *foutk* peuvent utiliser soit une chaîne de caractères contenant un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.

## Voir Aussi

*fiopen, fout, fouti, foutir*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# fprintks

fprintks — Semblable à printks mais imprime dans un fichier.

## Description

Semblable à *printks* mais imprime dans un fichier.

## Syntaxe

```
fprintks "filename", "string", [, kval1] [, kval2] [...]
```

## Initialisation

*"filename"* -- nom du fichier de sortie.

*"string"* -- la chaîne de texte à imprimer. Peut contenir jusqu'à 8192 caractères et doit être entre guillemets.

## Exécution

*kval1*, *kval2*, ... (facultatif) -- Les valeurs de taux-k à imprimer. Elle sont spécifiées dans « *string* » avec les spécificateurs de format du C standard (%f, %d, etc.) dans l'ordre donné.

*fprintks* est semblable à l'opcode *printks* sauf qu'il imprime dans un fichier et qu'il n'a pas de paramètre de taux-i. Pour plus d'information sur le formatage de la sortie, prière de consulter la documentation de *printks*.

## Exemples

Voici un exemple de l'opcode *fprintks*. Il utilise le fichier *fprintks.csd* [examples/fprintks.csd].

### Exemple 207. Exemple de l'opcode fprintks.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fprintks.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a score generator example.
```

```
instr 1
; K-rate stuff.
kstart init 0
kdur linrand 10
kpitch linrand 8

; Printing to to a file called "my.sco".
fprintks "my.sco", "i1\\t%2.2f\\t%2.2f\\t%2.2f\\n", kstart, kdur, 4+kpitch

knext linrand 1
kstart = kstart + knext
endin

</CsInstruments>
<CsScore>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play Instrument #1.
i 1 0 0.001

</CsScore>
</CsoundSynthesizer>
```

Cet exemple générera un fichier nommé « my.sco ». Il contiendra des lignes comme celles-ci :

```
i1      0.00      3.94      10.26
i1      0.20      3.35      6.22
i1      0.67      3.65      11.33
i1      1.31      1.42      4.13
```

Voici un exemple de l'opcode fprintks, qui convertit un fichier MIDI standard en partition Csound. Il utilise le fichier *fprintks-2.csd* [examples/fprintks-2.csd].

## Exemple 208. Exemple de l'opcode fprintks pour convertir un fichier MIDI en partition Csound.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
; -odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-n -Fmidichn_advanced.mid
;Don't write audio ouput to disk and use the file midichn_advanced.mid as MIDI input
</CsOptions>
<CsInstruments>

sr      = 48000
ksmps   = 16
nchnls  = 2

;Example by Jonathan Murphy 2007

; assign all midi events to instr 1000
massign 0, 1000
pgmassign 0, 1000

instr 1000

ktim timeinstd

kst, kch, kd1, kd2 midiin
if (kst != 0) then
; p4 = MIDI event type p5 = channel p6= data1 p7= data2
fprintks "MIDI2cs.sco", "i1\\t%f\\t%f\\t%d\\t%d\\t%d\\t%d\\n", ktim, 1/kr, kst, kch, kd1,
endif
endin
```

```
</CsInstruments>
<CsScore>
i1000 0 10000
e
</CsScore>
</CsoundSynthesizer>
```

Cet exemple générera un fichier nommé « MIDI2cs.sco » contenant des évènements i correspondant au fichier MIDI.

Voici un exemple avancé de l'opcode fprintks, qui génère une partition pour Csound. Il utilise le fichier *scogen-2.csd* [examples/scogen.csd].

### Exemple 209. Exemple de l'opcode fprintks pour créer un générateur de fichier de partition Csound au moyen de Csound.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
; -odac       -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-n
;Don't write audio ouput to disk
</CsOptions>
<CsInstruments>
;=====
;          scogen.csd          by: Matt Ingalls
;
;   a "port" of sorts
;   of the old "mills" score generator (scogen)
;
;   this instrument creates a schottstaedt.sco file
;   to be used with the schottstaedt.orc file
;
;   as long as you dont save schottstaedt.orc as a .csd
;   file, you should be able to keep it open in MacCsound
;   and render each newly generated .sco file.
;
;=====

gScoName = "/Users/matt/Desktop/schottstaedt.sco"      ; the name of the file to be generated

sr      =    100      ; this defines our temporal resolution,
                    ; an sr of 100 means we will generate p2 and p3 values
                    ; to the nearest 1/100th of a second

ksmps =    1          ; set kr=sr so we can do everything at k-rate

; some print opcodes
opcode PrintInteger, 0, k
    kval    xin
    fprintks    gScoName, "%d", kval
endop

opcode PrintFloat, 0, k
    kval    xin
    fprintks    gScoName, "%f", kval
endop

opcode PrintTab, 0, 0
    fprintks    gScoName, "%n"
endop

opcode PrintReturn, 0, 0
    fprintks    gScoName, "%r"
endop

; recursively calling opcode to handle all the optional parameters
opcode ProcessAdditionalPfields, 0, ikio
```

```

iPtable, kndx, iNumPfields, iPfield xin

; additional pfields start at 5, we use a default 0 to identify the first call
iPfield = (iPfield == 0 ? 5 : iPfield)

if (iPfield > iNumPfields) goto endloop
; find our tables
iMinTable table 2*iPfield-1, iPtable
iMaxTable table 2*iPfield, iPtable

; get values from our tables
kMin tablei kndx, iMinTable
kMax tablei kndx, iMaxTable

; find a random value in the range and write it to the score
fprintks gScoName, "%t%f", kMin + rnd(kMax-kMin)

; recursively call for any additional pfields.
ProcessAdditionalPfields iPtable, kndx, iNumPfields, iPfield + 1
endloop:

endop

/* =====
Generate a gesture of i-statements

p2 = start of the gesture
p3 = duration of the gesture
p4 = number of a function that contains a list of all
function table numbers used to define the
pfield random distribution
p5 = scale generated p4 values according to density (0=off, 1=on) [todo]
p6 = let durations overlap gesture duration (0=off, 1=on) [todo]
p7 = seed for random number generator seed [todo]
=====
*/
instr Gesture

; initialize
iResolution = 1/sr

kNextStart init p2
kCurrentTime init p2

iNumPfields table 0, p4
iInstrMinTable table 1, p4
iInstrMaxTable table 2, p4
iDensityMinTable table 3, p4
iDensityMaxTable table 4, p4
iDurMinTable table 5, p4
iDurMaxTable table 6, p4
iAmpMinTable table 7, p4
iAmpMaxTable table 8, p4

; check to make sure there is enough data
print iNumPfields
if iNumPfields < 4 then
    prints "%dError: At least 4 p-fields (8 functions) need to be specified.%n", iNumPfields
    turnoff
endif

; initial comment
fprints gScoName, "%!Generated Gesture from %f to %f seconds%n %!%t%twith a p-max of %d%n%n", p2

; k-rate stuff
if (kCurrentTime >= kNextStart) then ; write a new note!

    kndx = (kCurrentTime-p2)/p3

    ; get the required pfield ranges
    kInstMin tablei kndx, iInstrMinTable
    kInstMax tablei kndx, iInstrMaxTable
    kDensMin tablei kndx, iDensityMinTable
    kDensMax tablei kndx, iDensityMaxTable
    kDurMin tablei kndx, iDurMinTable
    kDurMax tablei kndx, iDurMaxTable
    kAmpMin tablei kndx, iAmpMinTable
    kAmpMax tablei kndx, iAmpMaxTable

    ; find random values for all our required parametrs and print the i-statement

```



```

fprintks gScoName, "i%d%t%f%t%f%t%f", kInstMin + rnd(kInstMax-kInstMin), kNextStart, kDurMin +
; now any additional pfields
ProcessAdditionalPfields p4, kndx, iNumPfields

PrintReturn

; calculate next starttime
kDensity = kDensMin + rnd(kDensMax-kDensMin)
if (kDensity < iResolution) then
    kDensity = iResolution
endif
kNextStart = kNextStart + kDensity
endif

kCurrentTime = kCurrentTime + iResolution
endin

</CsInstruments>
<CsScore>
/*
=====
scogen.sco

this csound module generates a score file
you specify a gesture of notes by giving
the "gesture" instrument a number to a
(negative) gen2 table.

this table stores numbers to pairs of functions.
each function-pair represents a range (min-max)
of randomness for every pfield for the notes to
be generated.
=====
*/

; common tables for pfield ranges
f100 0 2 -7 0 2 0 ; static 0
f101 0 2 -7 1 2 1 ; static 1
f102 0 2 -7 0 2 1 ; ramp 0->1
f103 0 2 -7 1 2 0 ; ramp 1->0
f105 0 2 -7 10 2 10 ; static 10
f106 0 2 -7 .1 2 .1 ; static .1

; specific pfield ranges
f10 0 2 -7 .8 2 .01 ; density
f11 0 2 -7 8 2 4 ; pitchmin
f12 0 2 -7 8 2 12 ; pitchmax

;=== table containing the function numbers used for all the p-field distributions
;
; p1 - table number
; p2 - time table is instantiated
; p3 - size of table (must be >= p5!)
; p4 - gen# (should be = -2)
; p5 - number of pfields of each note to be generated
; p6 - table number of the function representing the minimum possible note number (p1) of a genera
; p7 - table number of the function representing the maximum possible note number (p1) of a genera
; p8 - table number of the function representing the minimum possible noteon-to-noteon time (p2)
; p9 - table number of the function representing the maximum possible noteon-to-noteon time (p2)
; p10 - table number of the function representing the minimum possible duration (p3) of a genera
; p11 - table number of the function representing the maximum possible duration (p3) of a genera
; p12 - table number of the function representing the maximum possible amplitude (p4) of a genera
; p13 - table number of the function representing the maximum possible amplitude (p5) of a genera
; p14,p16.. - table number of the function representing the minimum possible value for additional
; p15,p17.. - table number of the function representing the maximum possible value for additional

; siz 2 #pds p1min p1max p2min p2max p3min p3max p4min p4max p5min p5max p6
f1 0 32 -2 6 101 101 10 10 101 105 100 106 11 12 100 101

;gesture definitions
; start dur pTble scale overlap seed
i"Gesture" 0 60 1 ;todo-->0 0 123
</CsScore>
</CsoundSynthesizer>

```

Cet exemple générera un fichier nommé « schottstaedt.sco » que l'on peut utiliser comme partition avec *schottstaedt.orc* [examples/schottstaedt.orc]

## Voir Aussi

*printks*

## Crédits

Auteur : Matt Ingalls  
Janvier 2003

# fprints

fprints — Semblable à prints mais imprime dans un fichier.

## Description

Semblable à *prints* mais imprime dans un fichier.

## Syntaxe

```
fprints "filename", "string" [, ival1] [, ival2] [...]
```

## Initialisation

*"filename"* -- nom du fichier de sortie.

*"string"* -- la chaîne de texte à imprimer. Peut contenir jusqu'à 8192 caractères et doit être entre guillemets.

*ival1*, *ival2*, ... (facultatif) -- Les valeurs de taux-i à imprimer. Elle sont spécifiées dans « *string* » avec les spécificateurs de format du C standard (%f, %d, etc.) dans l'ordre donné.

## Exécution

*fprints* est semblable à l'opcode *prints* sauf qu'il imprime dans un fichier. Pour plus d'information sur le formatage de la sortie, prière de consulter la documentation de *prints*.

## Exemples

Voici un exemple de l'opcode fprints. Il utilise le fichier *fprints.csd* [examples/fprints.csd].

### Exemple 210. Exemple de l'opcode fprints.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fprints.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a score generator example.
instr 1
```

```
; Print to the file "my.sco".
fprints "my.sco", "%!Generated score by ma++\\n \\n"
endin

</CsInstruments>
<CsScore>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play Instrument #1.
i 1 0 0.001

</CsScore>
</CsoundSynthesizer>
```

Cet exemple générera un fichier nommé « my.sco ». Il contiendra cette ligne :

```
!Generated score by ma++
```

## Voir Aussi

*prints*

## Crédits

Auteur : Matt Ingalls  
Janvier 2003

# frac

frac — Retourne la partie fractionnaire d'un nombre décimal.

## Description

Retourne la partie fractionnaire de  $x$ .

## Syntaxe

**frac**( $x$ ) (arguments de taux-i ou de taux-k ; fonctionne aussi au taux-a dans Csound5)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode frac. Il utilise le fichier *frac.csd* [examples/frac.csd].

### Exemple 211. Exemple de l'opcode frac.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o frac.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
i1 = 16 / 5
i2 = frac(i1)

    print i2
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i2 = 0.200
```

## Voir Aussi

*abs, exp, int, log, log10, i, sqrt*

## Crédits

Exemple écrit par Kevin Conder.

# freeverb

freeverb — Opcode version of Jezar's Freeverb

## Description

freeverb is a stereo reverb unit based on Jezar's public domain C++ sources, composed of eight parallel comb filters on both channels, followed by four allpass units in series. The filters on the right channel are slightly detuned compared to the left channel in order to create a stereo effect.

## Syntax

```
aoutL, aoutR freeverb ainL, ainR, kRoomSize, kHFDamp[, iSRate[, iSkip]]
```

## Initialization

*iSRate* (optional, defaults to 44100): adjusts the reverb parameters for use with the specified sample rate (this will affect the length of the delay lines in samples, and, as of the latest CVS version, the high frequency attenuation). Only integer multiples of 44100 will reproduce the original character of the reverb exactly, so it may be useful to set this to 44100 or 88200 for an orchestra sample rate of 48000 or 96000 Hz, respectively. While *iSRate* is normally expected to be close to the orchestra sample rate, different settings may be useful for special effects.

*iSkip* (optional, defaults to zero): if non-zero, initialization of the opcode will be skipped, whenever possible.

## Performance

*ainL*, *ainR* -- input signals; usually both are the same, but different inputs can be used for special effect



### Note

It is recommended to process the input signal(s) with the denorm opcode in order to avoid denormalized numbers which could significantly increase CPU usage in some cases

*aoutL*, *aoutR* -- output signals for left and right channel

*kRoomSize* (range: 0 to 1) -- controls the length of the reverb, a higher value means longer reverb. Settings above 1 may make the opcode unstable.

*kHFDamp* (range: 0 to 1): high frequency attenuation; a value of zero means all frequencies decay at the same rate, while higher settings will result in a faster decay of the high frequency range.

## Examples

Here is an example of the *freeverb* opcode. It uses the file *freeverb.csd* [examples/freeverb.csd].

### Exemple 212. An example of the freeverb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o freeverb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr           = 44100
ksmps       = 32
nchnls      = 2
0dbfs       = 1

instr 1
a1          vco2 0.75, 440, 10
kfrq        port 100, 0.008, 20000
a1          butterlp a1, kfrq
a2          linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1          = a1 * a2
a1          denorm a1
aL, aR      freeverb a1, a1, 0.9, 0.35, sr, 0
outs a1 + aL, a1 + aR
endin

</CsInstruments>
<CsScore>
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Istvan Varga  
2005



# ftchnls

ftchnls — Retourne le nombre de canaux dans un table de fonction en mémoire.

## Description

Retourne le nombre de canaux dans un table de fonction en mémoire.

## Syntaxe

`ftchnls(x)` (arg de taux-i seulement)

## Exécution

Retourne le nombre de canaux d'une table *GEN01*, déterminé par l'en-tête du fichier d'origine. Si le fichier d'origine n'a pas d'en-tête ou si la table n'a pas été créée par *GEN01*, *ftchnls* retourne -1.

## Exemples

Voici un exemple de l'opcode *ftchnls*. Il utilise les fichiers *ftchnls.csd* [examples/ftchnls.csd] et *mary.wav* [examples/mary.wav].

### Exemple 213. Exemple de l'opcode *ftchnls*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftchnls.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the number of channels in Table #1.
ichnls = ftchnls(1)
print ichnls
endin

</CsInstruments>
<CsScore>

; Table #1: Use an audio file, Csound will determine its size.
f 1 0 0 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
i 1 0 1
```

e

```
</CsScore>  
</CsoundSynthesizer>
```

Comme le fichier son « mary.wav » est monophonique (1 canal), la sortie comprendra une ligne comme celle-ci :

```
instr 1:  ichnls = 1.000
```

## Voir Aussi

*flen, filptim, ftsr, nsamp*

## Crédits

Auteur : Chris McCormick  
Perth, Australie  
Décembre 2001

Exemple écrit par Kevin Conder.

# ftconv

ftconv — Convolution multi-canaux à faible latence, utilisant une table de fonction pour la réponse impulsionnelle.

## Description

Convolution multi-canaux à faible latence, utilisant une table de fonction pour la réponse impulsionnelle. L'algorithme divise la réponse impulsionnelle en morceaux dont la longueur est déterminée par le paramètre *iplen*, et retarde et mixe ces morceaux de façon à ce que la réponse impulsionnelle originale soit reconstruite sans lacunes. Le délai de la sortie (latence) est de *iplen* échantillons et ne dépend pas du taux de contrôle, à la différence des autres opcodes de convolution.

## Syntaxe

```
a1[, a2[, a3[, ... a8]]] ftconv ain, ift, iplen[, iskip samples \
    [, iirlen[, iskipinit]]]
```

## Initialisation

*ift* -- numéro de la ftable source. La table doit contenir les données audio des différents canaux, entrelacées, avec un nombre de canaux égal au nombre de variables de sortie ((a1, a2, etc.). On peut créer une table entrelacée à partir d'un ensemble de tables mono avec *GEN52*.

*iplen* -- longueur des morceaux de réponse impulsionnelle en trames d'échantillon ; doit être une puissance entière de deux. Avec de faibles valeurs on aura un délai de sortie plus court, mais au prix d'une utilisation accrue du CPU.

*iskipsamples* (facultatif, 0 par défaut) -- nombre de trames d'échantillon à ignorer au début de la table. Utile pour les réponses de réverbération possédant un délai initial. Si ce délai n'est pas inférieur à *iplen* échantillons, en affectant à *iskipsamples* la même valeur que *iplen*, on éliminera toute latence supplémentaire de *ftconv*.

*iirlen* (facultatif) -- longueur totale de la réponse impulsionnelle en trames d'échantillon. Par défaut, on utilise toutes les données de la table (sans le point de garde).

*iskipinit* (facultatif, 0 par défaut) -- s'il a une valeur non nulle, l'initialisation est ignorée lorsque cela est possible sans causer d'erreur.

## Exécution

*ain* -- signal d'entrée

*a1 ... a8* -- signaux de sortie.

## Exemple

Voici un exemple de l'opcode ftconv. Il utilise le fichier *ftconv.csd* [examples/ftconv.csd].

### Exemple 214. Exemple de l'opcode ftconv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftconv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr        = 48000
ksmps     = 32
nchnls    = 2
0dbfs     = 1

garvb     init 0
gaW       init 0
gaX       init 0
gaY       init 0

itmp      ftgen 1, 0, 64, -2, 2, 40, -1, -1, -1, 123, \
           1, 13.000, 0.05, 0.85, 20000.0, 0.0, 0.50, 2, \
           1, 2.000, 0.05, 0.85, 20000.0, 0.0, 0.25, 2, \
           1, 16.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
           1, 9.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
           1, 12.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
           1, 8.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2

itmp      ftgen 2, 0, 262144, -2, 0
spat3dt 2, -0.2, 1, 0, 1, 1, 2, 0.005

itmp      ftgen 3, 0, 262144, -52, 3, 2, 0, 4, 2, 1, 4, 2, 2, 4

instr 1

al        vco2 1, 440, 10
kfrq      port 100, 0.008, 20000
al        butterlp al, kfrq
a2        linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
al        = al * a2 * 2
denorm    al
vincr     garvb, al
aw, ax, ay, az spat3di al, p4, p5, p6, 1, 1, 2
vincr     gaW, aw
vincr     gaX, ax
vincr     gaY, ay

endin

instr 2

denorm    garvb
; skip as many samples as possible without truncating the IR
arW, arX, arY ftconv garvb, 3, 2048, 2048, (65536 - 2048)
aW        = gaW + arW
aX        = gaX + arX
aY        = gaY + arY
garvb     = 0
gaW       = 0
gaX       = 0
gaY       = 0

aWre, aWim hilbert aW
aXre, aXim hilbert aX
aYre, aYim hilbert aY
aWXr      = 0.0928*aXre + 0.4699*aWre
aWXiYr    = 0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aL        = aWXr + aWXiYr
aR        = aWXr - aWXiYr

outs      aL, aR

endin

</CsInstruments>
<CsScore>
```

```
i 1 0 0.5 0.0 2.0 -0.8
i 1 1 0.5 1.4 1.4 -0.6
i 1 2 0.5 2.0 0.0 -0.4
i 1 3 0.5 1.4 -1.4 -0.2
i 1 4 0.5 0.0 -2.0 0.0
i 1 5 0.5 -1.4 -1.4 0.2
i 1 6 0.5 -2.0 0.0 0.4
i 1 7 0.5 -1.4 1.4 0.6
i 1 8 0.5 0.0 2.0 0.8
i 2 0 10
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*pconvolve, convolve, dconv.*

## Crédits

Auteur : Istvan Varga  
2005

# ftfree

ftfree — Efface une table de fonction.

## Description

Efface une table de fonction.

## Syntaxe

```
ftfree ifno, iwhen
```

## Initialisation

*ifno* -- le numéro de la table à effacer.

*iwhen* -- s'il vaut zéro, la table est effacée pendant la période d'initialisation ; sinon le numéro de table est enregistré pour que celle-ci soit effacée lors de la désactivation de la note.

## Crédits

Auteurs : Steven Yi, Istvan Varga  
2005

# ftgen

ftgen — Génère une table de fonction de partition depuis l'orchestre.

## Description

Génère une table de fonction de partition depuis l'orchestre.

## Syntaxe

```
gir ftgen ifn, itime, isize, igen, iarga [, iargb ] [...]
```

## Initialisation

*gir* -- un numéro de table soit demandé soit assigné automatiquement supérieur à 100.

*ifn* -- numéro de table demandé. Si *ifn* vaut zéro, le numéro est assigné automatiquement et sa valeur est placée dans *gir*. Toute autre valeur est utilisée comme le numéro de la table.

*itime* -- est ignoré, mais il correspond cependant au p2 de l'*instruction f* de partition.

*isize* -- taille de la table. Correspond au p3 de l'*instruction f* de partition.

*igen* -- routine *GEN* de la table de fonction. Correspond au p4 de l'*instruction f* de partition.

*iarga*, *iargb*, ... -- arguments de la table de fonction. Correspondent de p5 à pn de l'*instruction f* de partition.

## Exécution

Equivalent à la génération de table dans la partition au moyen de l'*instruction f*.



### Note

A l'origine, Csound était conçu pour ne supporter que les tables dont la taille était une puissance de deux. Bien que ceci ait changé dans les versions récentes (on peut utiliser n'importe quelle taille en donnant un nombre négatif), de nombreux opcodes ne les accepteront pas.



### Avertissement

Bien que Csound ne proteste pas si *ftgen* est utilisé à l'intérieur d'une paire d'instructions *instr-endin*, ce n'est pas une utilisation attendue ni supportée, et celle-ci doit être traitée avec prudence car elle a des effets globaux. En particulier, une taille différente conduit habituellement à une réallocation de la table, ce qui peut causer un plantage ou un comportement erratique.

## Exmples

Voici un exemple de l'opcode *ftgen*. Il utilise le fichier *ftgen.csd* [examples/ftgen.csd].

### Exemple 215. Exemple de l'opcode ftgen.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftgen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a sine wave using the GEN10 routine.
gitemp ftgen 1, 0, 16384, 10, 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ; Use Table #1.
  ifn = 1

  al oscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode ftgen. Il utilise le fichier *ftgen-2.csd* [examples/ftgen-2.csd].

### Exemple 216. Exemple de l'opcode ftgen.

Cet exemple interroge un fichier sur sa longueur afin de créer une f-table de la taille appropriée.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftgen-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

  sr      = 48000
  ksmps   = 16
  nchnls  = 2
```



*;Example by Jonathan Murphy 2007*

```

0dbfs      = 1

  instr 1

  Sfile      = "beats.wav"

  ilen      filelen  Sfile ; Find length
  isr       filesr   Sfile ; Find sample rate

  isamps     = ilen * isr ; Total number of samples
  isize      init    1

loop:
  isize      = isize * 2
  ; Loop until isize is greater than number of samples
if (isize < isamps) igoto loop

  itab       ftgen    0, 0, isize, 1, Sfile, 0, 0, 0
               print    isize
               print    isamps

  turnoff
  endin

</CsInstruments>
<CsScore>
il 0 10
e
</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*Routines GEN, ftgentmp*

## Crédits

Auteur : Barry L. Vercoe  
M.I.T., Cambridge, Mass  
1997

Exemple écrit par Kevin Conder.

Avertissement ajouté en Avril 2002 par Rasmus Ekman

# ftgenonce

ftgenonce — Génère une table de fonction depuis la définition d'un instrument, sans duplication de données.

## Description

Génère une table de fonction depuis la définition d'un instrument, sans duplication de données.

## Syntaxe

```
ifno ftgenonce ipldummy, ip2dummy, isize, igen, iarga, iargb, ...
```

## Initialisation

*ifno* -- un numéro de table automatiquement assigné supérieur à 100.

*ip1* -- ignoré.

*ip2dummy* -- ignoré.

*isize* -- taille de la table. Correspond au p3 de l'*instruction f* de partition.

*igen* -- routine *GEN* de la table de fonction. Correspond au p4 de l'*instruction f* de partition.

*iarga, iargb, ...* -- arguments de la table de fonction. Correspondent de p5 à pn de l'*instruction f* de partition.



### Note

A l'origine, Csound était conçu pour ne supporter que les tables dont la taille était une puissance de deux. Bien que ceci ait changé dans les versions récentes (on peut utiliser n'importe quelle taille en donnant un nombre négatif), de nombreux opcodes ne les accepteront pas.

## Exécution

L'opcode *ftgenonce* est conçu pour simplifier l'écriture des définitions d'instrument qui peuvent être réutilisées dans différents orchestres par un simple *#include* qui les insère dans un instrument. Il n'y a pas besoin de définir les tables de fonctions dans la partition ou dans l'en-tête de l'orchestre.

L'opcode *ftgenonce* est semblable à *ftgentmp*, et il a les mêmes arguments. Cependant, les tables de fonction ne sont ni dupliquées ni effacées. Au lieu de cela, tous les arguments de l'opcode sont concaténés pour former la clé d'accès à une table qui pointe vers le numéro de la table de fonction. Ainsi, chaque requête à *ftgenonce* avec les mêmes arguments reçoit la même instance des données de la table de fonction. Chaque changement de valeur d'un des arguments de *ftgenonce* provoque la création d'une nouvelle table de fonction.

## Crédits

Auteur : Michael Gogins  
2009

# ftgentmp

ftgentmp — Génère une table de fonction de partition depuis l'orchestre, qui est effacée à la fin de la note.

## Description

Génère une table de fonction de partition depuis l'orchestre, qui est facultativement effacée à la fin de la note.

## Syntaxe

```
ifno ftgentmp ip1, ip2dummy, isize, igen, iarga, iargb, ...
```

## Initialisation

*ifno* -- un numéro de table soit demandé soit assigné automatiquement supérieur à 100.

*ip1* -- le numéro de la table à générer ou 0 si le numéro doit être assigné, auquel cas la table est effacée à la fin de la période d'activation de la note.

*ip2dummy* -- ignoré.

*isize* -- taille de la table. Correspond au p3 de l'*instruction f* de partition.

*igen* -- routine *GEN* de la table de fonction. Correspond au p4 de l'*instruction f* de partition.

*iarga, iargb, ...* -- arguments de la table de fonction. Correspondent de p5 à pn de l'*instruction f* de partition.



### Note

A l'origine, Csound était conçu pour ne supporter que les tables dont la taille était une puissance de deux. Bien que ceci ait changé dans les versions récentes (on peut utiliser n'importe quelle taille en donnant un nombre négatif), de nombreux opcodes ne les accepteront pas.

## Exemples

Voici un exemple de l'opcode ftgentmp. Il utilise le fichier *ftgentmp.csd* [examples/ftgentmp.csd].

### Exemple 217. Exemple de l'opcode ftgentmp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
  
</CsOptions>  
<CsInstruments>
```

```

sr = 44100
ksmps = 128
nchnls = 2
odbfs = 1

instr 1
ifno ftgentmp 0, 0, 512, 10, 1
print ifno
endin

instr 2
print ftlen(p4)
endin

</CsInstruments>
<CsScore>
i 1 0 10
i 2 2 1 101
i 1 5 10
i 2 7 1 102
i 2 12 1 101
i 2 17 1 102
e
</CsScore>
</CsoundSynthesizer>

```

La sortie de ce csd montre que les tables sont détruites après la fin des instances d'instrument qui les ont créées, provoquant une erreur d'initialisation si les tables sont demandées.

```

SECTION 1:
new alloc for instr 1:
ftable 101:
instr 1: ifno = 101.000
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 0.00000 0.00000
new alloc for instr 2:
instr 2: #i0 = 512.000
B 2.000 .. 5.000 T 5.001 TT 5.001 M: 0.00000 0.00000
new alloc for instr 1:
ftable 102:
instr 1: ifno = 102.000
B 5.000 .. 7.000 T 7.001 TT 7.001 M: 0.00000 0.00000
instr 2: #i0 = 512.000
B 7.000 .. 12.000 T 11.999 TT 11.999 M: 0.00000 0.00000
INIT ERROR in instr 2: Invalid ftable no. 101.000000
#i0 ftlen.i p4
instr 2: #i0 = -1.000
B 12.000 - note deleted. i2 had 1 init errors
B 12.000 .. 17.000 T 17.000 TT 17.000 M: 0.00000 0.00000
INIT ERROR in instr 2: Invalid ftable no. 102.000000
#i0 ftlen.i p4
instr 2: #i0 = -1.000
B 17.000 - note deleted. i2 had 1 init errors

```

## Crédits

Auteur : Istvan Varga  
2005

# ftlen

ftlen — Retourne la taille d'une table de fonction en mémoire.

## Description

Retourne la taille d'une table de fonction en mémoire.

## Syntaxe

**ftlen**(x) (arg de taux-i seulement)

## Exécution

Retourne la taille (nombre de points, en excluant le point de garde) de la table de fonction numéro *x*. Bien que la plupart des unités faisant référence à une table en mémoire prennent automatiquement en compte sa taille (ce qui permet d'avoir des tables de longueur arbitraire), cette fonction retourne la taille actuelle en cas de besoin. Noter que *ftlen* retourne toujours une puissance de deux, ce qui veut dire que le point de garde de la table de fonction (voir *Instruction f*) n'est pas compris. A partir de Csound 3.53, *ftlen* travaille avec les tables de fonction différées (voir *GEN01*).

*ftlen* diffère de *nsamp* en ce sens que *nsamp* donne le nombre de trames d'échantillon chargées, tandis que *ftlen* donne le nombre total d'échantillons sans le point de garde. Par exemple, avec un fichier son stéréo de 10000 échantillons, *ftlen()* retournera 19999 (c'est-à-dire un total de 20000 échantillons mono, en excluant le point de garde), mais *nsamp()* retournera 10000.

## Exemples

Voici un exemple de l'opcode *ftlen*. Il utilise les fichiers *ftlen.csd* [examples/ftlen.csd] et *mary.wav* [examples/mary.wav].

### Exemple 218. Exemple de l'opcode *ftlen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftlen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the size of Table #1.
; The size will be the number of points excluding the guard point.
```

```
ilen = ftlen(1)
print ilen
endin

</CsInstruments>
<CsScore>

; Table #1: Use an audio file, Csound will determine its size.
f 1 0 0 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Le fichier audio « mary.wav » contient 154390 échantillons. L'opcode *ftlen* retourne une taille de 154389 échantillons car il réserve un point pour le point de garde. Sas sortie comprendra une ligne comme celle-ci :

```
instr 1:  ilen = 154389.000
```

## Voir Aussi

*ftchnls, filptim, ftsr, nsamp*

## Crédits

Auteur : Barry L. Vercoe  
MIT  
Cambridge, Massachussetts  
1997

Exemple écrit par Kevin Conder.

# ftload

ftload — Charge depuis un fichier un ensemble de tables préalablement allouées.

## Description

## Syntaxe

```
ftload "filename", iflag, ifn1 [, ifn2] [...]
```

## Initialisation

*"filename"* -- Une chaîne de caractères entre guillemets contenant le nom du fichier à charger.

*iflag* -- Type du fichier à charger (0 = binaire, différent de 0 = fichier texte).

*ifn1, ifn2, ...* -- Numéros des tables à charger.

## Exécution

*ftload* charge une liste de tables depuis un fichier. (Les tables doivent avoir été déjà allouées.) Le format du fichier peut être binaire ou texte.



### Avertissement

Le format du fichier n'est pas compatible avec un fichier WAV et l'ordre des octets (endianness) n'est pas sûr.

## Exemples

Voir l'exemple pour *ftsav*.

## Voir Aussi

*ftloadk, ftsavek, ftsave*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.21



# ftloadk

ftloadk — Charge depuis un fichier un ensemble de tables préalablement allouées.

## Description

Charge depuis un fichier un ensemble de tables préalablement allouées.

## Syntaxe

```
ftloadk "filename", ktrig, iflag, ifn1 [, ifn2] [...]
```

## Initialisation

*"filename"* -- Une chaîne de caractères entre guillemets contenant le nom du fichier à charger.

*iflag* -- Type du fichier à charger (0 = binaire, différent de 0 = fichier texte).

*ifn1*, *ifn2*, ... -- Numéros des tables à charger.

## Exécution

*ktrig* -- Le signal de déclenchement. Le fichier est chargé chaque fois que ce signal est différent de zéro.

*ftloadk* charge une liste de tables depuis un fichier. (Les tables doivent avoir été déjà allouées.) Le format du fichier peut être binaire ou texte. A la différence de *ftload*, l'opération de chargement peut-être répétée de nombreuses fois pendant la même note en utilisant un signal de déclenchement.



### Avertissement

Le format du fichier n'est pas compatible avec un fichier WAV et l'ordre des octets (endianness) n'est pas sûr.

## Voir Aussi

*ftload*, *ftsavek*, *ftsav*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.21

# ftlptim

ftlptim — Retourne la date du début de boucle d'une table de fonction en mémoire.

## Description

Retourne la date du début de boucle d'une table de fonction en mémoire.

## Syntaxe

`ftlptim(x)` (arg de taux-i seulement)

## Exécution

Retourne la date du début de boucle (en secondes) de la table de fonction numéro *x*. La valeur retournée est la durée de l'attaque et du decay directement enregistrés avant le segment de boucle. Retourne zéro (et un message d'avertissement) si l'échantillon ne contient pas de points de boucle.

## Exemples

Voici un exemple de l'opcode `ftlptim`. Il utilise les fichiers *ftlptim.csd* [examples/ftlptim.csd] et *mary.wav* [examples/mary.wav].

### Exemple 219. Exemple de l'opcode `ftlptim`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftlptim.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the loop-segment start time in Table #1.
itim = ftlptim(1)
print itim
endin

</CsInstruments>
<CsScore>

; Table #1: Use an audio file, Csound will determine its size.
f 1 0 0 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
```

```
i 1 0 1  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Comme le fichier audio « mary.wav » n'a pas de boucle, la sortie comprendra des lignes comme celles-ci :

```
WARNING: non-looping sample  
instr 1: itim = 0.000
```

## Voir Aussi

*ftchnls, flen, ftsr, nsamp*

## Crédits

Auteur : Barry L. Vercoe  
MIT  
Cambridge, Massachussetts  
1997

Exemple écrit par Kevin Conder.

# ftmorf

ftmorf — Morphing entre plusieurs ftables données dans une liste.

## Description

Utilise un index dans une table de numéros de ftable pour faire un morphing entre les tables voisines dans la liste. La fonction résultante est écrite dans la table référencée par *iresfn* à chaque cycle-k.

## Syntaxe

```
ftmorf kftndx, iftfn, iresfn
```

## Initialisation

*iftfn* -- la fonction de ftable. Ses valeurs doivent être des numéros de ftable pré-existantes.

*iresfn* -- numéro de table de la fonction résultante.

Toutes les tables référencées dans *iftfn* doivent avoir la même longueur que *iresfn*.

## Exécution

*kftndx* -- l'index dans la table *iftfn*.

Si *iftfn* contient (6, 4, 6, 8, 7, 4):

- *kftndx*=4 écrira le contenu de f7 dans *iresfn*.
- *kftndx*=4.5 écrira la moyenne des contenus de f7 et de f4 dans *iresfn*.

## Exemples

Voici un exemple de l'opcode ftmorf. Il utilise le fichier *ftmorf.csd* [examples/ftmorf.csd].

### Exemple 220. Exemple de l'opcode ftmorf.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftmorf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```

kr = 4410
ksmps = 10
nchnls = 1

instr 1
kndx line 0, p3, 7
      ftmorf kndx, 1, 2
asig oscili 30000, 440, 2
      out asig
endin

</CsInstruments>
<CsScore>

f1 0 8 -2 3 4 5 6 7 8 9 10
f2 0 1024 10 1 /*contents of f2 dont matter */
f3 0 1024 10 1
f4 0 1024 10 0 1
f5 0 1024 10 0 0 1
f6 0 1024 10 0 0 0 1
f7 0 1024 10 0 0 0 0 1
f8 0 1024 10 0 0 0 0 0 1
f9 0 1024 10 0 0 0 0 0 0 1
f10 0 1024 10 1 1 1 1 1 1 1 1

i1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Author : William « Pete » Moss  
 Université du Texas à Austin  
 Austin, Texas USA  
 Janvier 2002

Nouveau dans la version 4.18

# ftsave

ftsave — Sauvegarde dans un fichier un ensemble de tables préalablement allouées.

## Description

Sauvegarde dans un fichier un ensemble de tables préalablement allouées.

## Syntaxe

```
ftsave "filename", iflag, ifn1 [, ifn2] [...]
```

## Initialisation

*"filename"* -- Une chaîne de caractères entre guillemets contenant le nom du fichier à sauvegarder.

*iflag* -- Type du fichier à sauvegarder (0 = binaire, différent de 0 = fichier texte).

*ifn1*, *ifn2*, ... -- Numéros des tables à sauvegarder.

## Exécution

*ftsave* sauvegarde une liste de tables dans un fichier. Le format du fichier peut être binaire ou texte.



### Avertissement

Le format du fichier n'est pas compatible avec un fichier WAV et l'ordre des octets (endianness) n'est pas sûr.

## Exemples

Voici un exemple de l'opcode ftsave. Il utilise le fichier *ftsave.csd* [examples/ftsave.csd].

### Exemple 221. Exemple de l'opcode ftsave.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftsave.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```

; Table #1, make a sine wave using the GEN10 routine.
gitmp1 ftgen 1, 0, 32768, 10, 1
; Table #2, create an empty table.
gitmp2 ftgen 2, 0, 32768, 7, 0, 32768, 0

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 20000
  kcps = 440
  ; Use Table #1.
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin

; Instrument #2 - Load Table #1 into Table #2.
instr 2
  ; Save Table #1 to a file called "table1.ftsave".
  ftsave "table1.ftsave", 0, 1

  ; Load the "table1.ftsave" file into Table #2.
  ftload "table1.ftsave", 0, 2

  kamp = 20000
  kcps = 440
  ; Use Table #2, it should contain Table #1's sine wave now.
  ifn = 2

  a1 oscil kamp, kcps, ifn
  out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
; Play Instrument #2 for 1 second.
i 2 2 1
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir Aussi

*ftloadk, ftload, ftsavek*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.21

# ftsavek

ftsavek — Sauvegarde dans un fichier un ensemble de tables préalablement allouées.

## Description

Sauvegarde dans un fichier un ensemble de tables préalablement allouées.

## Syntaxe

```
ftsavek "filename", ktrig, iflag, ifn1 [, ifn2] [...]
```

## Initialisation

*"filename"* -- Une chaîne de caractères entre guillemets contenant le nom du fichier à sauvegarder.

*iflag* -- Type du fichier à sauvegarder (0 = binaire, différent de 0 = fichier texte).

*ifn1*, *ifn2*, ... -- Numéros des tables à sauvegarder.

## Exécution

*ktrig* -- Le signal de déclenchement. Le fichier est sauvegardé chaque fois que ce signal est différent de zéro.

*ftsavek* sauvegarde une liste de tables dans un fichier. Le format du fichier peut être binaire ou texte. A la différence de *ftsave*, l'opération de sauvegarde peut-être répétée de nombreuses fois pendant la même note en utilisant un signal de déclenchement.



### Avertissement

Le format du fichier n'est pas compatible avec un fichier WAV et l'ordre des octets (endianness) n'est pas sûr.

## Voir Aussi

*ftloadk*, *ftload*, *ftsave*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.21



# ftsr

ftsr — Retourne le taux d'échantillonnage d'une table de fonction en mémoire.

## Description

Retourne le taux d'échantillonnage d'une table de fonction en mémoire.

## Syntaxe

**ftsr**(x) (arg de taux-i seulement)

## Exécution

Retourne le taux d'échantillonnage d'une table générée par *GEN01*. Le taux d'échantillonnage est déterminé à partir de l'en-tête du fichier original. Si ce dernier n'a pas d'en-tête ou si la table n'a pas été créée avec *GEN01*, *ftsr* retourne 0. Nouveau dans la version 3.49 de Csound.

## Exemples

Voici un exemple de l'opcode ftsr. Il utilise les fichiers *ftsr.csd* [examples/ftsr.csd] et *mary.wav* [examples/mary.wav].

### Exemple 222. Exemple de l'opcode ftsr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the sampling rate of Table #1.
isr = ftsr(1)
print isr
endin

</CsInstruments>
<CsScore>

; Table #1: Use an audio file.
f 1 0 262144 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
```

```
i 1 0 1  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Comme le fichier son « mary.wav » utilise un taux d'échantillonnage de 44.1 Khz, la sortie comprendra une ligne comme celle-ci :

```
instr 1:  isr = 44100.000
```

## Voir Aussi

*ftchnls, ftlen, ftlptim, nsamp*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Octobre 1998

Exemple écrit par Kevin Conder.

# gain

gain — Ajuste l'amplitude d'un signal audio en fonction d'une valeur efficace.

## Description

Ajuste l'amplitude d'un signal audio en fonction d'une valeur efficace.

## Syntaxe

```
ares gain asig, krms [, ihp] [, iskip]
```

## Initialisation

*ihp* (facultatif, 10 par défaut) -- point à mi-puissance (en Hz) d'un d'un filtre passe-bas interne spécial. La valeur par défaut est 10.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace de données interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio en entrée

*gain* effectue une modification d'amplitude de *asig* de sorte que la sortie *ares* ait pour valeur effice *krms*. *rms* et *gain* utilisés conjointement (et avec des valeurs de *ihp* correspondantes) produiront le même effet que *balance*.

## Exemples

Voici un exemple de l'opcode gain. Il utilise le fichier *gain.csd* [examples/gain.csd].

### Exemple 223. Exemple de l'opcode gain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gain.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
```

```
asrc buzz 30000, 440, sr/440, 1 ; band-limited pulse train.
a1 reson asrc, 1000, 100 ; Sent through
a2 reson a1, 3000, 500 ; 2 filters
krms rms asrc ; then balanced
afin gain a2, krms ; with source
out afin

endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*balance, rms*

# gainslider

**gainslider** — Une implémentation de courbe de gain logarithmique qui est semblable à l'objet **gainslider~** de Cycling 74 Max / MSP.

## Description

Cet opcode sert à être multiplié par un signal audio pour donner la même impression qu'avec une console de mixage. Il n'y a pas de limites dans le code source si bien que l'on peut par exemple donner des valeurs supérieures à 127 pour obtenir un signal audio plus fort mais avec un risque d'écroulement.

## Syntaxe

```
kout gainslider kindex
```

## Exécution

*kindex* -- Valeur d'indice. Intervalle nominal de 0 à 127. Par exemple un intervalle de 0 à 152 donnera un intervalle de -# à +18,0 dB.

*kout* -- Sortie pondérée.

## Exemples

Voici un exemple de l'opcode **gainslider**. Il utilise le fichier *gainslider.csd* [examples/gainslider.csd].

### Exemple 224. Exemple de l'opcode **gainslider**.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   Silent
-odac         -iadc      -d      ;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 100
nchnls = 2

instr 1 ; gainslider test

; uncomment for realtime midi
;kmod ctrl17 1, 1, 0, 127

; uncomment for non realtime
km0d phasor 1/10
kmod scale km0d, 127, 0

kout gainslider kmod

printks "kmod = %f kout = %f\\n", 0.1, kmod, kout

aout diskin2 "fox.wav", 1, 0, 1
```

```
aout = aout*kout
    outs aout, aout
    endin

</CsInstruments>
<CsScore>
i1 0 30
e
</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*scale, logcurve, expcurve*

## Crédits

Auteur : David Akbari  
Octobre  
2006

# gauss

gauss — Générateur de nombres aléatoires de distribution gaussienne.

## Description

Générateur de nombres aléatoires de distribution gaussienne. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares gauss krange
```

```
ires gauss krange
```

```
kres gauss krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (*-krange* à *+krange*). Produit des nombres positifs et négatifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode gauss. Il utilise le fichier *gauss.csd* [examples/gauss.csd].

### Exemple 225. Exemple de l'opcode gauss.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; -o gauss.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a random number between -1 and 1.
```

```

; krange = 1

i1 gauss 1

print i1
endin

; Instrument #2.
instr 2
; Generate a random number between -1 and 1.
; krange = 1

seed 0

i1 gauss 1

print i1
endin

</CsInstruments>
<CsScore>
; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1: i1 = 0.252
```

## Voir Aussi

*seed, betarand, bexprnd, cauchy, exprand, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.

Existait dans la version 3.30



# gbuzz

gbuzz — La sortie est un ensemble de partiels cosinus en relation harmonique.

## Description

La sortie est un ensemble de partiels cosinus en relation harmonique.

## Syntaxe

```
ares gbuzz xamp, xcps, knh, klh, kmul, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de table d'une fonction stockée contenant une onde cosinus. Une grande table d'au moins 8192 points est recommandée.

*iphs* (facultatif, par défaut 0) -- phase initiale de la fréquence fondamentale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

## Exécution

Les unités *buzz* génèrent un ensemble additif de partiels cosinus en relation harmonique de fréquence fondamentale *xcps*, et dont les amplitudes sont pondérées de telle façon que la crête de leur somme égale *xamp*. Le choix et l'importance des partiels sont déterminés par les paramètres de contrôle suivants :

*knh* -- nombre total d'harmoniques demandés. Si *knh* est négatif, sa valeur absolue est utilisée. Si *knh* vaut zéro, une valeur de 1 est utilisée.

*klh* -- harmonique présent le plus bas. Peut être positif, nul ou négatif. Dans *gbuzz* l'ensemble de partiels peut commencer à n'importe quel numéro de partiel et se complète vers le haut ; si *klh* est négatif, tous les partiels en-dessous de zéro seront repliés comme des partiels positifs sans changement de phase (car le cosinus est une fonction paire), et s'ajouteront de façon constructive aux partiels positifs de l'ensemble.

*kmul* -- spécifie la raison de la série des coefficients d'amplitude. C'est une série entière : si le *klh*ème partiel a pour coefficient *A*, le (*klh* + *n*)ème partiel aura pour coefficient  $A * (kmul ** n)$ , c'est-à-dire que les valeurs d'intensité dessinent une courbe exponentielle. *kmul* peut être positif, nul ou négatif, et n'est pas restreint aux valeurs entières.

*buzz* et *gbuzz* sont utiles comme sources de son complexe dans la synthèse soustractive. *buzz* est un cas particulier du plus général *gbuzz* dans lequel *klh* = *kmul* = 1 ; il produit ainsi un ensemble de *knh* harmoniques de même importance, commençant avec le fondamental. (C'est un train d'impulsions à bande de fréquence limitée ; si les partiels vont jusqu'à la fréquence de Nyquist, c'est-à-dire  $knh = \text{int} (sr / 2 / \text{fréq. fondamentale})$ , le résultat est un train d'impulsions réelles d'amplitude *xamp*.)

Bien que l'on puisse faire varier *knh* et *klh* durant l'exécution, leurs valeurs internes sont nécessairement entières ce qui peut provoquer des « pops » dûs à des discontinuités dans la sortie. Cependant, la variation de *kmul* durant l'exécution produit un bon effet. *gbuzz* peut être modulé en amplitude et/ou en fréquence soit par des signaux de contrôle soit par des signaux audio.

Nota Bene : cette unité a son pendant avec *GENII*, dans lequel le même ensemble de cosinus peut être

stocké dans une table de fonction qui sera lue par un oscillateur. Bien que plus efficace en termes de calcul, le train d'impulsions stocké a un contenu spectral fixe, non variable dans le temps comme celui décrit ci-dessus.

## Exemples

Voici un exemple de l'opcode `gbuzz`. Il utilise le fichier `gbuzz.csd` [examples/gbuzz.csd].

### Exemple 226. Exemple de l'opcode `gbuzz`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gbuzz.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 20000
  kcps = 440
  knh = 3
  klh = 2
  kmul = 0.7
  ifn = 1

  a1 gbuzz kamp, kcps, knh, klh, kmul, ifn
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a simple cosine waveform.
f 1 0 16384 11 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*`buzz`*

## Crédits

Exemple écrit par Kevin Conder.

Septembre 2003. Merci à Kanata Motohashi pour avoir corrigé les mentions du paramètre *kmul*.

# getcfg

getcfg — Return Csound settings.

## Description

Return various configuration settings in Svalue as a string at init time.

## Syntax

Svalue **getcfg** iopt

## Initialization

*iopt* -- The parameter to be returned, can be one of:

- 1: the maximum length of string variables in characters; this is at least the value of the `--max_str_len` command line option - 1
- 2: the input sound file name (-i), or empty if there is no input file
- 3: the output sound file name (-o), or empty if there is no output file
- 4: return "1" if real time audio input or output is being used, and "0" otherwise
- 5: return "1" if running in beat mode (-t command line option), and "0" otherwise
- 6: the host operating system name
- 7: return "1" if a callback function for the `chnrecv` and `chnsend` opcodes has been set, and "0" otherwise (which means these opcodes do nothing)

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# gogobel

gogobel — La sortie audio est un son tel que celui produit lorsque l'on frappe une cloche à vache.

## Description

La sortie audio est un son tel que celui produit lorsque l'on frappe une cloche à vache. Il s'agit d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares gogobel kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivfn
```

## Initialisation

*ihrd* -- la dureté de la baguette utilisée pour la frappe. On utilise un intervalle allant de 0 à 1. 0,5 est une valeur adéquate.

*ipos* -- le point d'impact sur le bloc, compris entre 0 et 1.

*imp* -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

*ivfn* -- forme du vibrato, habituellement une table sinus, créée par une fonction.

## Exécution

Une note est jouée sur une instrument de type cloche à vache, avec les arguments suivants.

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kvibf* -- Fréquence du vibrato en Hertz. L'intervalle conseillé va de 0 à 12.

*kvamp* -- Amplitude du vibrato.

## Exemples

Voici un exemple de l'opcode gogobel. Il utilise les fichiers *gogobel.csd* [examples/gogobel.csd] et *marmstk1.wav* [examples/marmstk1.wav]

### Exemple 227. Exemple de l'opcode gogobel.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in   No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gogobel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; kamp = 31129.60
; kfreq = 440
; ihrd = 0.5
; ipos = 0.561
; imp = 1
; kvibf = 6.0
; kvamp = 0.3
; ivfn = 2

a1 gogobel 31129.60, 440, 0.5, 0.561, 1, 6.0, 0.3, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, the "marmstk1.wav" audio file.
f 1 0 256 1 "marmstk1.wav" 0 0 0
; Table #2, a sine wave for the vibrato.
f 2 0 128 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# goto

goto — Transfère le contrôle à chaque passage.

## Description

Transfère le contrôle vers *label* à chaque passage. Combinaison de *igoto* et de *kgoto*

## Syntaxe

`goto label`

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.



### Note

Si l'on utilise *goto* en dehors d'une instruction *if* (comme dans : goto end), l'initialisation sera ignorée pour tout le code sauté par le *goto*. Si dans une exécution quelques opcodes ne sont pas initialisés, cela provoquera l'effacement de la note ou de l'évènement. Dans ce cas, il peut être préférable d'utiliser *kgoto* (comme dans : kgoto end).

## Exemples

Voici un exemple de l'opcode goto. Il utilise le fichier *goto.csd* [examples/goto.csd].

### Exemple 228. Exemple de l'opcode goto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o goto.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  a1 oscil 10000, 440, 1
  goto playit

; The goto will go to the playit label.
; It will skip any code in between like this comment.

playit:
  out a1
endin
```

```
</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*cggoto, cigoto, ckgoto, if, igoto, kgoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

Note ajoutée par Jim Aikin.



# grain

grain — Génère des textures de synthèse granulaire.

## Description

Génère des textures de synthèse granulaire.

## Syntaxe

```
ares grain xamp, xpitch, xdens, kampoff, kpitchoff, kgdur, igfn, \  
iwfn, imgdur [, igrnd]
```

## Initialisation

*igfn* -- numéro de la ftable de la forme d'onde du grain. Peut être une onde sinus ou un son échantillonné.

*iwfn* -- numéro de la ftable de l'enveloppe d'amplitude utilisée pour les grains (voir aussi *GEN20*).

*imgdur* -- durée maximum du grain en secondes. C'est la plus grande valeur que l'on peut affecter à *kgdur*.

*igrnd* (facultatif) -- s'il est non nul, le décalage aléatoire du grain est désactivé. Cela signifie que tous les grains commenceront à lire la table *igfn* depuis son début. S'il vaut zéro (par défaut), les grains commenceront leur lecture dans la table *igfn* à partir de positions aléatoires.

## Exécution

*xamp* -- amplitude de chaque grain.

*xpitch* -- hauteur du grain. Pour utiliser la fréquence originale du son en entrée, on se sert de la formule :

$$\text{sndsr} / \text{ftlen}(\text{igfn})$$

où *sndsr* est le taux d'échantillonnage original du son *igfn*.

*xdens* -- densité des grains mesurée en grains par seconde. Si elle est constante la sortie sera une synthèse granulaire synchrone, très semblable à *fof*. Si *xdens* a une composante aléatoire (comme du bruit ajouté), alors le résultat ressemblera plus à une synthèse granulaire asynchrone.

*kampoff* -- déviation d'amplitude maximale par rapport à *xamp*. Cela signifie que l'amplitude maximale possible pour un grain est *xamp + kampoff* et l'amplitude minimale est *xamp*. Si *kampoff* est nul alors il n'y a pas d'amplitude aléatoire pour chaque grain.

*kpitchoff* -- déviation de hauteur maximale par rapport à *xpitch* en Hz. Semblable à *kampoff*.

*kgdur* -- durée du grain en secondes. Sa valeur maximale doit être déclarée dans *imgdur*. Si *kgdur* dépasse *imgdur* en un point, sa valeur sera tronquée à celle de *imgdur*.

Le générateur *grain* est principalement basé sur les travaux et les écrits de Barry Truax et de Curtis Roads.

## Exemples

Cet exemple génère une texture avec des grains de plus en plus courts, une amplitude de plus en plus large et une dispersion de hauteur. Il utilise les fichiers *grain.csd* [examples/grain.csd] et *mary.wav* [examples/mary.wav].

### Exemple 229. Exemple de l'opcode grain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o grain.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1
  insnd = 10
  ibasfrq = 44100 / ftlen(insnd)  ; Use original sample rate of insnd file

  kamp   expseg 220, p3/2, 600, p3/2, 220
  kpitch line ibasfrq, p3, ibasfrq * .8
  kdens  line 600, p3, 200
  kaoff  line 0, p3, 5000
  kpoff  line 0, p3, ibasfrq * .5
  kgdur  line .4, p3, .1
  imaxgdur = .5

  ar grain kamp, kpitch, kdens, kaoff, kpoff, kgdur, insnd, 5, imaxgdur, 0.0
  out ar
endin

</CsInstruments>
<CsScore>

f5 0 512 20 2          ; Hanning window
f10 0 262144 1 "mary.wav" 0 0 0
i1 0 6
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Paris Smaragdis  
MIT  
Mai 1997

# grain2

grain2 — Générateur de textures par synthèse granulaire facile à utiliser.

## Description

Génère des textures par synthèse granulaire. *grain2* est plus simple à utiliser, mais *grain3* offre plus de contrôle.

## Syntaxe

```
ares grain2 kcps, kfmd, kgdur, iovrlp, kfn, iwfn [, irpow] \  
      [, iseed] [, imode]
```

## Initialisation

*iovrlp* -- (constant) nombre de grains se chevauchant.

*iwfn* -- table de fonction contenant la forme d'onde d'une fenêtre (utiliser GEN20 pour calculer *iwfn*).

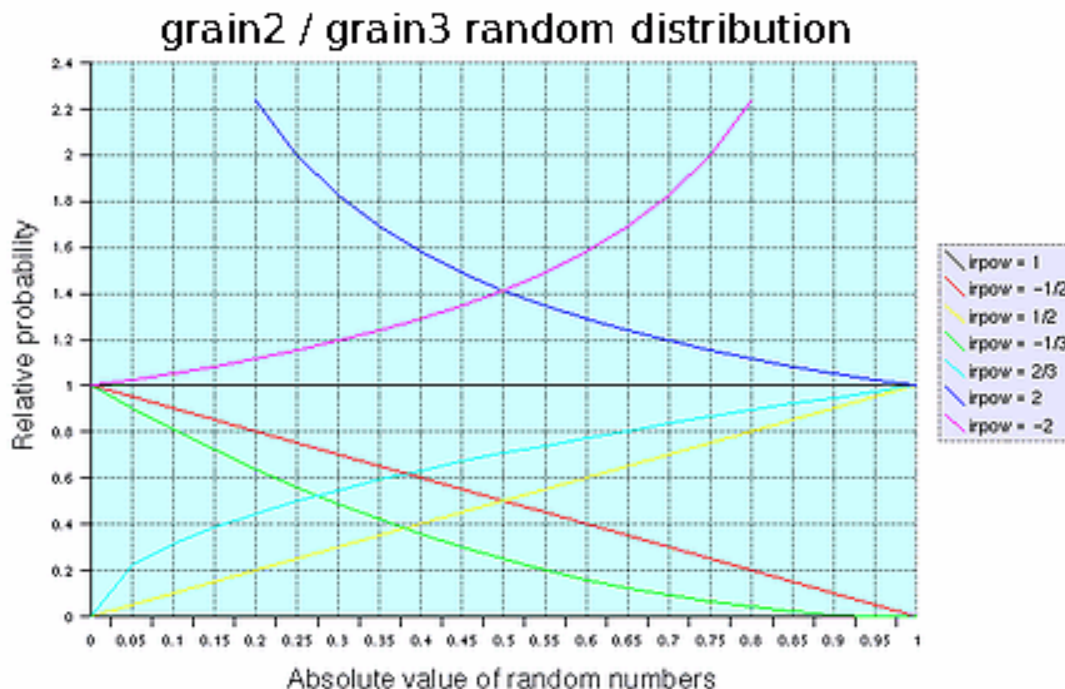
*irpow* (facultatif, par défaut 0) -- cette valeur contrôle la variation de la distribution de la fréquence du grain. Si *irpow* est positif, la distribution aléatoire ( $x$  est compris entre -1 et 1) est

$\text{abs}(x) ^ ((1 / \text{irpow}) - 1) ;$

pour des valeurs négatives de *irpow*, elle est

$(1 - \text{abs}(x)) ^ ((-1 / \text{irpow}) - 1)$

En fixant *irpow* à -1, 0, ou 1 on obtiendra une distribution uniforme (dont le calcul est plus rapide). L'image ci-dessous montre quelques exemples pour *irpow*. La valeur par défaut de *irpow* est 0.



Un graphique des distributions pour différentes valeurs de *irpow*.

*iseed* (facultatif, par défaut 0) -- valeur de la graine du générateur de nombres aléatoires (entier positif compris entre 1 et 2147483646 ( $2^{31} - 2$ )). Une valeur nulle ou négative force la graine à prendre la valeur de l'horloge de l'ordinateur (c'est le comportement par défaut).

*imode* (facultatif, par défaut 0) -- somme de valeurs prises parmi les suivantes :

- 8 : forme d'onde de la fenêtre avec interpolation (plus lent).
- 4 : pas d'interpolation pour la forme d'onde des grains (rapide, mais de moindre qualité).
- 2 : la fréquence des grains est modifiée continuellement par *kcps* et *kfmd* (par défaut, chaque grain garde la fréquence avec laquelle il a démarré). Avec des taux de contrôle élevés, ceci peut ralentir le processus.
- 1 : ignorer l'initialisation.

## Exécution

*ares* -- signal de sortie.

*kcps* -- fréquence du grain en Hz.

*kfmd* -- variation aléatoire (bipolaire) de la fréquence du grain en Hz.

*kgdur* -- durée du grain en secondes. *kgdur* contrôle aussi la durée des grains déjà actifs (en fait la vitesse à laquelle la fonction fenêtre est lue). Ce comportement ne dépend pas des indicateurs positionnés dans *imode*.

*kfn* -- table de fonction contenant la forme d'onde du grain. Le numéro de table peut changer au taux-k

(on peut ainsi choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter le repliement).



## Note

*grain2* utilise en interne le même générateur aléatoire que *rnd31*. Il est ainsi recommandé de lire également sa *documentation*.

## Exemples

Voici un exemple de l'opcode *grain2*. Il utilise le fichier *grain2.csd* [examples/grain2.csd].

### Exemple 230. Exemple de l'opcode *grain2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o grain2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 48000
kr = 750
ksmps = 64
nchnls = 2

/* square wave */
i_ ftgen 1, 0, 4096, 7, 1, 2048, 1, 0, -1, 2048, -1
/* window */
i_ ftgen 2, 0, 16384, 7, 0, 4096, 1, 4096, 0.3333, 8192, 0
/* sine wave */
i_ ftgen 3, 0, 1024, 10, 1
/* room parameters */
i_ ftgen 7, 0, 64, -2, 4, 50, -1, -1, -1, 11, \
      1, 26.833, 0.05, 0.85, 10000, 0.8, 0.5, 2, \
      1, 1.753, 0.05, 0.85, 5000, 0.8, 0.5, 2, \
      1, 39.451, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
      1, 33.503, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
      1, 36.151, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
      1, 29.633, 0.05, 0.85, 7000, 0.8, 0.5, 2

ga01 init 0

/* generate bandlimited square waves */

i0 = 0
loop1:
imaxh = sr / (2 * 440.0 * exp(log(2.0) * (i0 - 69) / 12))
i_ ftgen i0 + 256, 0, 4096, -30, 1, 1, imaxh
i0 = i0 + 1
if (i0 < 127.5) igoto loop1

instr 1

p3 = p3 + 0.2

/* note velocity */
iamp = 0.0039 + p5 * p5 / 16192
/* vibrato */
kcps oscili 1, 8, 3
kenv linseg 0, 0.05, 0, 0.1, 1, 1, 1
/* frequency */
kcps = (kcps * kenv * 0.01 + 1) * 440 * exp(log(2) * (p4 - 69) / 12)
```

```
/* grain ftable */
kfn = int(256 + 69 + 0.5 + 12 * log(kcps / 440) / log(2))
/* grain duration */
kgdur port 100, 0.1, 20
kgdur = kgdur / kcps

a1 grain2 kcps, kcps * 0.02, kgdur, 50, kfn, 2, -0.5, 22, 2
a1 butterlp a1, 3000
a2 grain2 kcps, kcps * 0.02, 4 / kcps, 50, kfn, 2, -0.5, 23, 2
a2 butterbp a2, 12000, 8000
a2 butterbp a2, 12000, 8000
aenv1 linseg 0, 0.01, 1, 1, 1
aenv2 linseg 3, 0.05, 1, 1, 1
aenv3 linseg 1, p3 - 0.2, 1, 0.07, 0, 1, 0

a1 = aenv1 * aenv3 * (a1 + a2 * 0.7 * aenv2)

ga01 = ga01 + a1 * 10000 * iamp

    endin

/* output instr */

    instr 81

i1 = 0.000001
aLl, aLh, aRl, aRh spat3di ga01 + i1*i1*i1*i1, 3.0, 4.0, 0.0, 0.5, 7, 4
ga01 = 0
aLl butterlp aLl, 800.0
aRl butterlp aRl, 800.0

    outs aLl + aLh, aRl + aRh

    endin

</CsInstruments>
<CsScore>

t 0 60

i 1 0.0 1.3 60 127
i 1 2.0 1.3 67 127
i 1 4.0 1.3 64 112
i 1 4.0 1.3 72 112

i 81 0 6.4

e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*grain3*

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

Mise à jour en avril 2002 par Istvan Varga

# grain3

grain3 — Générateur de textures par synthèse granulaire avec plus de contrôle.

## Description

Génère des textures par synthèse granulaire. *grain2* est plus simple à utiliser mais *grain3* offre plus de contrôle.

## Syntaxe

```
ares grain3 kcps, kphs, kfmd, kpmd, kgdur, kdens, imaxovr, kfn, iwfn, \  
      kfrpow, krpow [, iseed] [, imode]
```

## Initialisation

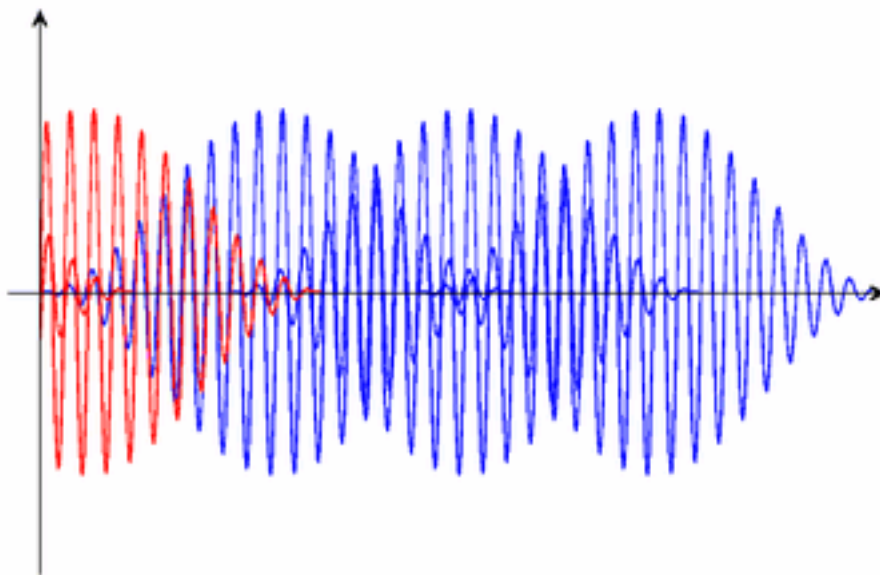
*imaxovr* -- nombre maximum de grains se chevauchant. Le nombre de chevauchements peut être calculé par (*kdens* \* *kgdur*) ; cependant, il peut être surestimé sans coût supplémentaire lors de l'exécution, et un simple chevauchement utilise (selon le système) de 16 à 32 octets en mémoire.

*iwfn* -- table de fonction contenant la forme d'onde d'une fenêtre (utiliser GEN20 pour calculer *iwfn*).

*iseed* (facultatif, par défaut 0) -- valeur de la graine du générateur de nombres aléatoires (entier positif compris entre 1 et 2147483646 ( $2^{31} - 2$ )). Une valeur nulle ou négative force la graine à prendre la valeur de l'horloge de l'ordinateur (c'est le comportement par défaut).

*imode* (facultatif, par défaut 0) -- somme de valeurs prises parmi les suivantes :

- *64* : synchronise la phase au démarrage des grains sur *kcps*.
- *32* : démarre tous les grains sur une position d'échantillon entière. Ceci peut être plus rapide dans certains cas, tout en rendant moins précis le déroulement temporel des enveloppes de grain.
- *16* : ne pas générer de grains ayant une date de démarrage inférieure à zéro. (Voir la figure ci-dessous ; cette option désactive les grains marqués en rouge sur l'image).
- *8* : forme d'onde de la fenêtre avec interpolation (plus lent).
- *4* : pas d'interpolation pour la forme d'onde des grains (rapide, mais de moindre qualité).
- *2* : la fréquence des grains est modifiée continuellement par *kcps* et *kfmd* (par défaut, chaque grain garde la fréquence avec laquelle il a démarré). Avec des taux de contrôle élevés, ceci peut ralentir le processus. Contrôle aussi la modulation de phase (*kphs*)
- *1* : ignorer l'initialisation.



Graphique montrant des grains avec une date de démarrage inférieure à zéro en rouge.

## Exécution

*ares* -- signal de sortie.

*kcps* -- fréquence du grain en Hz.

*kphs* -- phase du grain. C'est une position dans la forme d'onde du grain, exprimée comme une fraction (entre 0 et 1) de la longueur de la table.

*kfmd* -- variation aléatoire (bipolaire) de la fréquence du grain en Hz.

*kpmf* -- variation aléatoire (bipolaire) de la phase au démarrage.

*kgdur* -- durée du grain en secondes. *kgdur* contrôle aussi la durée des grains déjà actifs (en fait la vitesse à laquelle la fonction fenêtre est lue). Ce comportement ne dépend pas des indicateurs positionnés dans *imode*.

*kdens* -- nombre de grains par seconde.

*kfrpow* -- cette valeur contrôle la variation de la distribution de la fréquence du grain. Si *kfrpow* est positif, la distribution aléatoire (*x* est compris entre -1 et 1) est

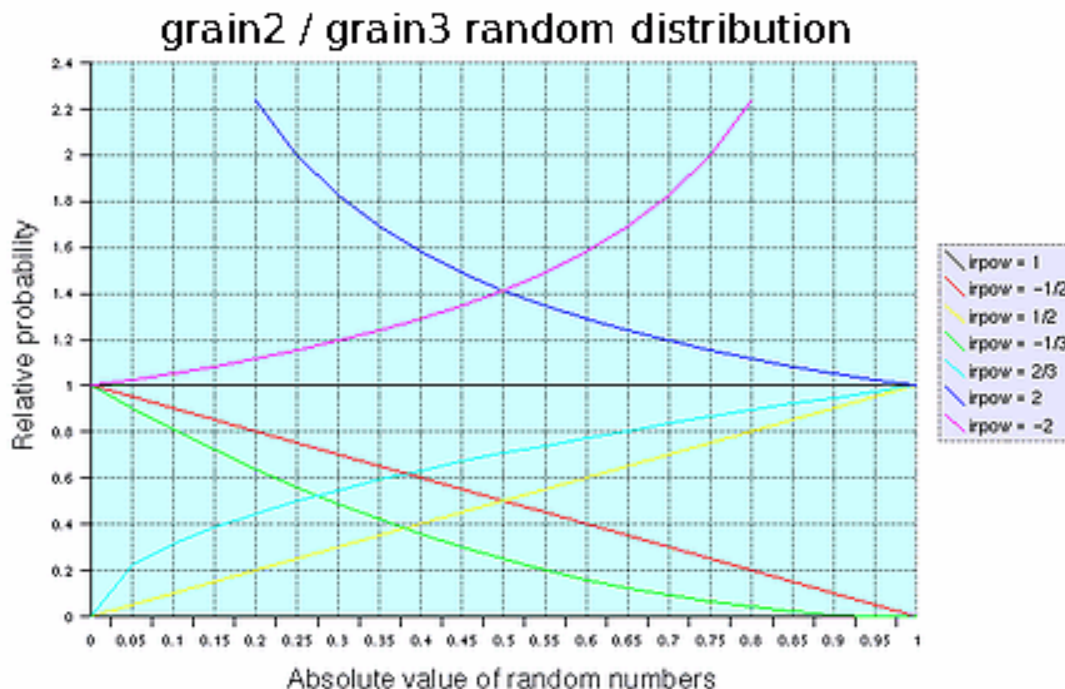
$\text{abs}(x)^{((1 / \text{kfrpow}) - 1)}$  ;

pour des valeurs négatives de *kfrpow*, elle est

$(1 - \text{abs}(x))^{((-1 / \text{kfrpow}) - 1)}$

En fixant *kfrpow* à -1, 0, ou 1 on obtiendra une distribution uniforme (dont le calcul est plus rapide). L'image ci-dessous montre quelques exemples pour *kfrpow*. La valeur par défaut de *kfrpow* est 0.





Un graphique des distributions pour différentes valeurs de *kfpow*.

*kfpow* -- variation de la distribution de phase aléatoire (voir *kfpow*). En fixant *kphs* et *kpmid* à 0.5, et *kfpow* à 0 on émule *grain2*.

*kfn* -- table de fonction contenant la forme d'onde du grain. Le numéro de table peut changer au taux-k (on peut ainsi choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter le repliement).



## Note

*grain3* utilise en interne le même générateur aléatoire que *rnd31*. Il est ainsi recommandé de lire également sa *documentation*.

## Exemples

Voici un exemple de l'opcode *grain3*. Il utilise le fichier *grain3.csd* [examples/grain3.csd].

### Exemple 231. Exemple de l'opcode *grain3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o grain3.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 48000
kr = 1000
ksmps = 48
nchnls = 1

/* Bartlett window */
itmp ftgen 1, 0, 16384, 20, 3, 1
/* sawtooth wave */
itmp ftgen 2, 0, 16384, 7, 1, 16384, -1
/* sine */
itmp ftgen 4, 0, 1024, 10, 1
/* window for "soft sync" with 1/32 overlap */
itmp ftgen 5, 0, 16384, 7, 0, 256, 1, 7936, 1, 256, 0, 7936, 0
/* generate bandlimited sawtooth waves */
itmp ftgen 3, 0, 4096, -30, 2, 1, 2048
icnt = 0
loop01:
; 100 tables for 8 octaves from 30 Hz
ifrq = 30 * exp(log(2) * 8 * icnt / 100)
itmp ftgen icnt + 100, 0, 4096, -30, 3, 1, sr / (2 * ifrq)
icnt = icnt + 1
if (icnt < 99.5) igoto loop01
/* convert frequency to table number */
#define FRQ2FNUM(xout'xcps'xbsfn) #

$xout = int(($xbsfn) + 0.5 + (100 / 8) * log(($xcps) / 30) / log(2))
$xout limit $xout, $xbsfn, $xbsfn + 99

#

/* instr 1: pulse width modulated grains */

instr 1

kfrq = 523.25 ; frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number
kfmd = kfrq * 0.02 ; random variation in frequency
kgdur = 0.2 ; grain duration
kdens = 200 ; density
iseed = 1 ; random seed

kphs oscili 0.45, 1, 4 ; phase

a1 grain3 kfrq, 0, kfmd, 0.5, kgdur, kdens, 100, \
kfnum, 1, -0.5, 0, iseed, 2
a2 grain3 kfrq, 0.5 + kphs, kfmd, 0.5, kgdur, kdens, 100, \
kfnum, 1, -0.5, 0, iseed, 2

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

out aenv * 2250 * (a1 - a2)

endin

/* instr 2: phase variation */

instr 2

kfrq = 220 ; frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number
kgdur = 0.2 ; grain duration
kdens = 200 ; density
iseed = 2 ; random seed

kprdst expon 0.5, p3, 0.02 ; distribution

a1 grain3 kfrq, 0.5, 0, 0.5, kgdur, kdens, 100, \
kfnum, 1, 0, -kprdst, iseed, 64

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

out aenv * 1500 * a1

endin

/* instr 3: "soft sync" */

```

```

instr 3

kdens = 130.8          ; base frequency
kgdur = 2 / kdens      ; grain duration

kfrq expon 880, p3, 220 ; oscillator frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number

a1 grain3 kfrq, 0, 0, 0, kgdur, kdens, 3, kfnum, 5, 0, 0, 0, 2
a2 grain3 kfrq, 0.667, 0, 0, kgdur, kdens, 3, kfnum, 5, 0, 0, 0, 2

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

out aenv * 10000 * (a1 - a2)

endin

</CsInstruments>
<CsScore>

t 0 60
i 1 0 3
i 2 4 3
i 3 8 3
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*grain2*

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

Mise à jour en avril 2002 par Istvan Varga

# granule

granule — Un générateur de texture par synthèse granulaire plus complexe.

## Description

Le générateur unitaire *granule* est plus complexe que *grain*, mais il ajoute de nouvelles possibilités.

*granule* est un générateur unitaire de Csound qui emploie une table d'onde en entrée pour produire une sortie audio par synthèse granulaire. Les données de la table d'onde peuvent être générées par n'importe laquelle des routines GEN telle que *GEN01* qui lit un fichier audio. On peut ainsi utiliser un son échantillonné comme source pour les grains. L'implémentation interne comprend jusqu'à 128 voix. Le nombre maximum de voix peut être augmenté en redéfinissant la variable MAXVOICE dans le fichier grain4.h. *granule* possède son propre générateur de nombres aléatoires pour produire toutes les fluctuations aléatoires des paramètres. Il comprend aussi une fonction de seuil pour scanner la table de fonction source lors de la phase d'initialisation. On peut ainsi facilement ignorer les passages de silence entre les phrases.

Les caractéristiques de la synthèse sont contrôlées par 22 paramètres. *xamp* est l'amplitude de la sortie et elle peut varier aussi bien au taux audio qu'au taux de contrôle.

## Syntaxe

```
ares granule xamp, ivoice, iratio, imode, ithd, ifn, ipshift, igskip, \  
    igskip_os, ilength, kgap, igap_os, kgsiz, igsiz_os, iatt, idec \  
    [, iseed] [, ipitch1] [, ipitch2] [, ipitch3] [, ipitch4] [, ifnenv]
```

## Initialisation

*ivoice* -- nombre de voix.

*iratio* -- rapport entre la vitesse du pointeur de lecture et le taux d'échantillonnage de la sortie, par exemple 0,5 donnera une vitesse de lecture moitié de la vitesse originale.

*imode* -- +1, le pointeur de lecture progresse en avant (direction naturelle du fichier source), -1, en arrière (direction opposée à la direction naturelle du fichier source), ou 0, pour une direction aléatoire.

*ithd* -- seuil ; lorsque le signal échantillonné dans la table est plus petit que *ithd*, il est ignoré.

*ifn* -- numéro de la table de fonction de la source sonore.

*ipshift* -- contrôle de la transposition. Si *ipshift* vaut 0, la hauteur sera fixée aléatoirement dans un ambitus d'une octave de part et d'autre de la hauteur de chaque grain. Si *ipshift* vaut 1, 2, 3 ou 4, on peut fixer jusqu'à quatre hauteurs différentes pour le nombre de voix défini dans *ivoice*. Les paramètres facultatifs *ipitch1*, *ipitch2*, *ipitch3* et *ipitch4* servent à quantifier les transpositions.

*igskip* -- décalage initial depuis le début de la table de fonction en sec.

*igskip\_os* -- fluctuation aléatoire du pointeur de lecture en sec, 0 signifiant pas de décalage.

*ilength* -- longueur de la partie de la table à utiliser à partir de *igskip* en sec.

*igap\_os* -- fluctuation aléatoire de l'écart en % de la taille de l'écart, 0 signifiant pas de décalage.

*igsize\_os* -- fluctuation aléatoire de la taille du grain en % de la taille du grain, 0 signifiant pas de décalage.

*iatt* -- attaque de l'enveloppe du grain en % de la taille du grain.

*idec* -- chute de l'enveloppe du grain en % de la taille du grain.

*iseed* (facultatif, par défaut 0,5) -- graine pour le générateur de nombre aléatoire.

*ipitch1*, *ipitch2*, *ipitch3*, *ipitch4* (facultatif, par défaut 1) -- paramètre de transposition, utilisé lorsque *ipshift* vaut 1, 2, 3 ou 4. La transposition est réalisée par une technique de pondération temporelle avec interpolation linéaire entre les points. La valeur par défaut de 1 signifie la hauteur originale.

*ifnenv* (facultatif, par défaut 0) -- numéro de la table de fonction utilisée pour générer la forme de l'enveloppe.

## Exécution

*xamp* -- amplitude.

*kgap* -- écart entre les grains en sec.

*kgsize* -- taille du grain en sec.

## Exemples

Voici un exemple de l'opcode *granule*. Il utilise les fichiers *granule.csd* [examples/granule.csd], et *mary.wav* [examples/mary.wav].

### Exemple 232. Exemple de l'opcode *granule*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o granule.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
instr 1
;
k1      linseg 0,0.5,1,(p3-p2-1),1,0.5,0
a1      granule p4*k1,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,\
        p16,p17,p18,p19,p20,p21,p22,p23,p24
a2      granule p4*k1,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,\
        p16,p17,p18,p19, p20+0.17,p21,p22,p23,p24
outs a1,a2
endin

</CsInstruments>
<CsScore>

; f statement read sound file mary.wav in the SFDIR
```

```

; directory into f-table 1
f1      0 262144 1 "mary.wav" 0 0 0
il      0 10 2000 64 0.5 0 0 1 4 0 0.005 5 0.01 50 0.02 50 30 30 0.39 \
      1 1.42 0.29 2
e

</CsScore>
</CsoundSynthesizer>

```

L'exemple ci-dessus lit un fichier son nommé *mary.wav* dans la table de fonction numéro 1 en gardant 262 144 échantillons. Il génère 10 secondes de sortie stéréo à partir de la table de fonction. Dans le fichier orchestre, tous les paramètres nécessaires au contrôle de la synthèse proviennent du fichier partition. Un générateur de fonction *linseg* est utilisé pour produire une enveloppe avec une attaque et une chute linéaires de 0,5 secondes. On obtient un effet stéréo par l'utilisation de différentes graines pour les deux appels de la fonction *granule*. Dans l'exemple, on ajoute 0,17 à p20 avant de le passer au second appel de *granule* pour s'assurer que toutes les fluctuations aléatoires seront différentes de celles du premier appel.

Voici la signification des paramètres dans le fichier partition :

Parameter	Interpreted As
p5 ( <i>ivoice</i> )	le nombre de voix est fixé à 64
p6 ( <i>iratio</i> )	fixé à 0,5, on lit la table d'onde deux fois moins vite que le taux de la sortie audio
p7 ( <i>imode</i> )	fixé à 0, le pointeur du grain ne se déplace qu'en avant
p8 ( <i>ithd</i> )	fixé à 0, pas de détection de seuil
p9 ( <i>ifn</i> )	fixé à 1, on utilise la table de fonction numéro 1
p10 ( <i>ipshift</i> )	fixé à 4, quatre hauteurs différentes seront générées
p11 ( <i>igskip</i> )	fixé à 0 et p12 ( <i>igskip_os</i> ) est fixé à 0,005, pas de décalage par rapport au début de table d'onde et on utilise une fluctuation aléatoire de 5 ms
p13 ( <i>ilength</i> )	fixé à 5, on n'utilise que 5 secondes de la table d'onde
p14 ( <i>kgap</i> )	fixé à 0,01 et p15 ( <i>igap_os</i> ) est fixé à 50, on utilise un écart de 10 ms avec une fluctuation aléatoire de 50%
p16 ( <i>kgsiz</i> )	fixé à 0,02 et p17 ( <i>igsize_os</i> ) est fixé à 50, la durée du grain est de 20 ms avec une fluctuation aléatoire de 50%
p18 ( <i>iatt</i> ) et p19 ( <i>idec</i> )	fixés à 30, on applique une attaque et une chute linéaires de 30% au grain
p20 ( <i>iseed</i> )	la graine pour le générateur de nombre aléatoire est fixée à 0,39
p21 - p24	les hauteurs sont fixées à 1, soit la hauteur originale, 1,42 soit une quinte plus haut, 0,29 soit une septième plus bas et enfin 2 soit une octave plus haut.

## Crédits

Auteur : Allan Lee

Belfast

1996

Nouveau dans la version 3.35

# guiro

guiro — Modèle semi-physique d'un son de guiro.

## Description

*guiro* est un modèle semi-physique d'un son de guiro. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares guiro kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] [, ifreq1]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 128.

*idamp* (facultatif) -- le facteur d'amortissement de l'instrument. *Inutilisé*.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2500.

*ifreq1* (facultatif) -- la première fréquence de résonance.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode *guiro*. Il utilise le fichier *guiro.csd* [examples/guiro.csd].

### Exemple 233. Exemple de l'opcode *guiro*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o guiro.wav -W ;; for file output any platform
</CsOptions>
```



```
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

    instr 01 ;example of a guiro
a1 guiro p4, 0.01
    out a1
    endin

</CsInstruments>
<CsScore>

i1 0 1 20000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*bamboo, dripwater, sleighbells, tambourine*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# harmon

harmon — Analyze an audio input and generate harmonizing voices in synchrony.

## Description

Analyze an audio input and generate harmonizing voices in synchrony.

## Syntax

```
ares harmon asig, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, imode, \  
    iminfrq, iprd
```

## Initialization

*imode* -- interpreting mode for the generating frequency inputs *kgenfreq1*, *kgenfreq2*. 0: input values are ratios with respect to the audio signal analyzed frequency. 1: input values are the actual requested frequencies in Hz.

*iminfrq* -- the lowest expected frequency (in Hz) of the audio input. This parameter determines how much of the input is saved for the running analysis, and sets a lower bound on the internal pitch tracker.

*iprd* -- period of analysis (in seconds). Since the internal pitch analysis can be time-consuming, the input is typically analyzed only each 20 to 50 milliseconds.

## Performance

*kestfrq* -- estimated frequency of the input.

*kmaxvar* -- the maximum variance (expects a value between 0 and 1).

*kgenfreq1* -- the first generated frequency.

*kgenfreq2* -- the second generated frequency.

This unit is a harmonizer, able to provide up to two additional voices with the same amplitude and spectrum as the input. The input analysis is assisted by two things: an input estimated frequency *kestfrq* (in Hz), and a fractional maximum variance *kmaxvar* about that estimate which serves to limit the size of the search. Once the real input frequency is determined, the most recent pulse shape is used to generate the other voices at their requested frequencies.

The three frequency inputs can be derived in various ways from a score file or MIDI source. The first is the expected pitch, with a variance parameter allowing for inflections or inaccuracies; if the expected pitch is zero the harmonizer will be silent. The second and third pitches control the output frequencies; if either is zero the harmonizer will output only the non-zero request; if both are zero the harmonizer will be silent. When the requested frequency is higher than the input, the process requires additional computation due to overlapped output pulses. This is currently limited for efficiency reasons, with the result that only one voice can be higher than the input at any one time.

This unit is useful for supplying a background chorus effect on demand, or for correcting the pitch of a faulty input vocal. There is essentially no delay between input and output. Output includes only the generated parts, and does not include the input.

## Examples

Here is an example of the `harmon` opcode. It uses the file `harmon.csd` [examples/harmon.csd].

### Exemple 234. Example of the `harmon` opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d            ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o harmon.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; The frequency of the base note.
inote = 440

; Generate the base note.
avco vco 20000, inote, 1

kestfrq = inote
kmaxvar = 0.4

; Calculate frequencies 3 semitones above and
; below the base note.
kgenfreq1 = inote * semitone(3)
kgenfreq2 = inote * semitone(-3)

imode = 1
iminfrq = inote - 200
iprd = 0.1

; Generate the harmony notes.
al harmon avco, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, \
    imode, iminfrq, iprd

out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsSoundSynthesizer>
```

## Credits

Author: Barry L. Vercoe  
M.I.T., Cambridge, Mass  
1997

New in version 3.47

Example written by Kevin Conder.

# harmon2

**harmon2** — Analyze an audio input and generate harmonizing voices in synchrony with formants preserved.

## Description

Generate harmonizing voices with formants preserved.

## Syntax

```
ares harmon2 asig, koct, kfrq1, kfrq2, icpsmode, ilowest[, ipolarity]
```

```
ares harmon3 asig, koct, kfrq1, \  
    kfrq2, kfrq3, icpsmode, ilowest[, ipolarity]
```

```
ares harmon4 asig, koct, kfrq1, \  
    kfrq2, kfrq3, kfrq4, icpsmode, ilowest[, ipolarity]
```

## Initialization

*icpsmode* -- interpreting mode for the generating frequency inputs *kfrq1*, *kfrq2*, *kfrq3* and *kfrq4*: 0: input values are ratios w.r.t. the cps equivalent of *koct*. 1: input values are the actual requested frequencies in cps.

*ilowest* -- owest value of the koct input for which harmonizing voices will be generated.

*ipolarity* -- polarity of asig input, 1 = positive glottal pulses, 0 = negative. Default is 1.

## Performance

**Harmon2**, **harmon3** and **harmon4** are high-performance harmonizers, able to provide up to four pitch-shifted copies of the input *asig* with spectral formants preserved. The pitch-shifting algorithm requires an accurate running estimate (*koct*, in decimal oct units) of the pitched content of *asig*, normally gained from an independent pitch tracker such as *specptrk*. The algorithm then isolates the most recent full pulse within *asig*, and uses this to generate the other voices at their required pulse rates.

If the frequency (or ratio) presented to *kfrq1*, *kfrq2*, *kfrq3* or *kfrq4* is zero, then no signal is generated for that voice. If any of them is non-zero, but the *koct* input is below the value *ilowest*, then that voice will output a direct copy of the input *asig*. As a consequence, the data arriving at the k-rate inputs can variously cause the generated voices to be turned on or off, to pass a direct copy of a non-voiced fricative source, or to harmonize the source according to some constructed algorithm. The transition from one mode to another is cross-faded, giving seamless alternating between voiced (harmonized) and non-voiced fricatives during spoken or sung input.

*harmon2*, *harmon3*, *harmon4* are especially matched to the output of *specptrk*. The latter generates pitch data in decimal octave format; it also emits its base value if no pitch is identified (as in fricative noise) and emits zero if the energy falls below a threshold, so that *harmon2*, *harmon3*, *harmon4* can be set to pass the direct signal in both cases. Of course, any other form of pitch estimation could also be used. Since pitch trackers usually incur a slight delay for accurate estimation (for *specptrk* the delay is printed by the spectrum unit), it is normal to delay the audio signal by the same amount so that *harmon2*, *harmon3*, *harmon4* can work from a fully concurrent estimate.

## Examples

Here is an example of the `harmon` opcode. It uses the file `harmon.csd` [examples/harmon.csd].

### Exemple 235. Example of the `harmon2` opcode.

```
a1,a2 ins                                ; get mic input
w1 spectrum      a1, .02, 7, 24, 12, 1, 3 ; and examine it
koct,kamp specptrk      w1, 1, 6.5, 9.5, 7.5, 10, 7, .7, 0, 3, 1
a3 delay      a1, .065      ; allow for ptrk delay
a4 harmon2      a3, koct, 1.25, 0.75, 0, 6.9 ; output a fixed 6-4 harmony
      outs      a3, a4      ; as well as the original
```

## Credits

Author: Barry L. Vercoe  
M.I.T., Cambridge, Mass  
2006

New in version 5.04

# hilbert

hilbert — A Hilbert transformer.

## Description

An IIR implementation of a Hilbert transformer.

## Syntax

```
ar1, ar2 hilbert asig
```

## Performance

*asig* -- input signal

*ar1* -- sine output of *asig*

*ar2* -- cosine output of *asig*

*hilbert* is an IIR filter based implementation of a broad-band 90 degree phase difference network. The input to *hilbert* is an audio signal, with a frequency range from 15 Hz to 15 kHz. The outputs of *hilbert* have an identical frequency response to the input (i.e. they sound the same), but the two outputs have a constant phase difference of 90 degrees, plus or minus some small amount of error, throughout the entire frequency range. The outputs are in quadrature.

*hilbert* is useful in the implementation of many digital signal processing techniques that require a signal in phase quadrature. *ar1* corresponds to the cosine output of *hilbert*, while *ar2* corresponds to the sine output. The two outputs have a constant phase difference throughout the audio range that corresponds to the phase relationship between cosine and sine waves.

Internally, *hilbert* is based on two parallel 6th-order allpass filters. Each allpass filter implements a phase lag that increases with frequency; the difference between the phase lags of the parallel allpass filters at any given point is approximately 90 degrees.

Unlike an FIR-based Hilbert transformer, the output of *hilbert* does not have a linear phase response. However, the IIR structure used in *hilbert* is far more efficient to compute, and the nonlinear phase response can be used in the creation of interesting audio effects, as in the second example below.

## Examples

The first example implements frequency shifting, or single sideband amplitude modulation. Frequency shifting is similar to ring modulation, except the upper and lower sidebands are separated into individual outputs. By using only one of the outputs, the input signal can be "detuned," where the harmonic components of the signal are shifted out of harmonic alignment with each other, e.g. a signal with harmonics at 100, 200, 300, 400 and 500 Hz, shifted up by 50 Hz, will have harmonics at 150, 250, 350, 450, and 550 Hz.

Here is the first example of the hilbert opcode. It uses the file *hilbert.csd* [examples/hilbert.csd], and *mary.wav* [examples/mary.wav].

**Exemple 236. Example of the hilbert opcode implementing frequency shifting.**

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o hilbert.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
  idur = p3
  ; Initial amount of frequency shift.
  ; It can be positive or negative.
  ibegshift = p4
  ; Final amount of frequency shift.
  ; It can be positive or negative.
  iendshift = p5

  ; A simple envelope for determining the
  ; amount of frequency shift.
  kfreq linseg ibegshift, idur, iendshift

  ; Use the sound of your choice.
  ain soundin "mary.wav"

  ; Phase quadrature output derived from input signal.
  areal, aimag hilbert ain

  ; Quadrature oscillator.
  asin oscili 1, kfreq, 1
  acos oscili 1, kfreq, 1, .25

  ; Use a trigonometric identity.
  ; See the references for further details.
  amod1 = areal * acos
  amod2 = aimag * asin

  ; Both sum and difference frequencies can be
  ; output at once.
  ; aupshift corresponds to the sum frequencies.
  aupshift = (amod1 - amod2) * 0.7
  ; adownshift corresponds to the difference frequencies.
  adownshift = (amod1 + amod2) * 0.7

  ; Notice that the adding of the two together is
  ; identical to the output of ring modulation.

  out aupshift
endin

</CsInstruments>
<CsScore>

; Sine table for quadrature oscillator.
f 1 0 16384 10 1

; Starting with no shift, ending with all
; frequencies shifted up by 200 Hz.
i 1 0 2 0 200

; Starting with no shift, ending with all
; frequencies shifted down by 200 Hz.
i 1 2 2 0 -200
e

</CsScore>
</CsoundSynthesizer>
```



The second example is a variation of the first, but with the output being fed back into the input. With very small shift amounts (i.e. between 0 and  $\pm 6$  Hz), the result is a sound that has been described as a « barberpole phaser » or « Shepard tone phase shifter. » Several notches appear in the spectrum, and are constantly swept in the direction opposite that of the shift, producing a filtering effect that is reminiscent of Risset's « endless glissando ».

Here is the second example of the hilbert opcode. It uses the file *hilbert\_barberpole.csd* [examples/hilbert\_barberpole.csd], and *mary.wav* [examples/mary.wav].

### Exemple 237. Example of the hilbert opcode sounding like a « barberpole phaser ».

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o hilbert_barberpole.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
; kr must equal sr for the barberpole effect to work.
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1
instr 1
  idur = p3
  ibegshift = p4
  iendshift = p5

  ; sawtooth wave, not bandlimited
  asaw phasor 100
  ; add offset to center phasor amplitude between -.5 and .5
  asaw = asaw - .5
  ; sawtooth wave, with amplitude of 10000
  ain = asaw * 20000

  ; The envelope of the frequency shift.
  kfreq linseg ibegshift, idur, iendshift

  ; Phase quadrature output derived from input signal.
  areal, aimag hilbert ain

  ; The quadrature oscillator.
  asin oscili 1, kfreq, 1
  acos oscili 1, kfreq, 1, .25

  ; Based on trigonometric identities.
  amod1 = areal * acos
  amod2 = aimag * asin

  ; Calculate the up-shift and down-shift.
  aupshift = (amod1 + amod2) * 0.7
  adownshift = (amod1 - amod2) * 0.7

  ; Mix in the original signal to achieve the barberpole effect.
  amix1 = aupshift + ain
  amix2 = adownshift + ain

  ; Make sure the output doesn't get louder than the original signal.
  aout1 balance amix1, ain
```

```
aout2 balance amix2, ain
outs aout1, aout2
endin

</CsInstruments>
<CsScore>

; Table 1: A sine wave for the quadrature oscillator.
f 1 0 16384 10 1

; The score.
; p4 = frequency shifter, starting frequency.
; p5 = frequency shifter, ending frequency.
i 1 0 6 -10 10
e

</CsScore>
</CsoundSynthesizer>
```

## Technical History

The use of phase-difference networks in frequency shifters was pioneered by Harald Bode.<sup>1</sup> Bode and Bob Moog provide an excellent description of the implementation and use of a frequency shifter in the analog realm in;<sup>2</sup> this would be an excellent first source for those that wish to explore the possibilities of single sideband modulation. Bernie Hutchins provides more applications of the frequency shifter, as well as a detailed technical analysis.<sup>3</sup> A recent paper by Scott Wardle<sup>4</sup> describes a digital implementation of a frequency shifter, as well as some unique applications.

## References

1. H. Bode, "Solid State Audio Frequency Spectrum Shifter." AES Preprint No. 395 (1965).
2. H. Bode and R.A. Moog, "A High-Accuracy Frequency Shifter for Professional Audio Applications." *Journal of the Audio Engineering Society*, July/August 1972, vol. 20, no. 6, p. 453.
3. B. Hutchins. *Musical Engineer's Handbook* (Ithaca, NY: Electronotes, 1975), ch. 6a.
4. S. Wardle, "A Hilbert-Transformer Frequency Shifter for Audio." Available online at <http://www.iua.upf.es/dafx98/papers/>.

## Credits

Author: Sean Costello  
Seattle, Washington  
1999

New in Csound version 3.55

The examples were updated April 2002. Thanks go to Sean Costello for fixing the barberpole example.

# hrtfer

hrtfer — Crée de l'audio 3D pour deux haut-parleurs.

## Description

La sortie audio en 3D est binaurale (casque stéréo).

## Syntaxe

```
aleft, aright hrtfer asig, kaz, kelev, « HRTFcompact »
```

## Initialisation

*kAz* -- valeur d'azimut en degrés. Les valeurs positives représentent les positions à droite, les valeurs négatives les positions à gauche.

*kElev* -- valeur d'élévation en degrés. Les valeurs positives représentent les positions au-dessus de l'horizontale, les valeurs négatives les positions sous l'horizontale.

Actuellement, le seul fichier qui peut être utilisé avec *hrtfer* est *HRTFcompact* [exemples/HRTFcompact]. Il doit être passé à l'opcode en dernier argument entre guillemets comme ci-dessus.

On peut aussi obtenir *HRTFcompact* par ftp anonyme depuis :  
`ftp://ftp.cs.bath.ac.uk/pub/dream/utilities/Analysis/HRTFcompact`

## Exécution

Ces générateurs unitaires placent un signal d'entrée mono dans un espace 3D virtuel autour de l'auditeur en faisant une convolution entre l'entrée et les données HRTF appropriées spécifiées par les valeurs d'azimut et d'élévation de l'opcode. *hrtfer* accepte que ces valeurs soient de taux-k, ce qui permet une spatialisation dynamique. *hrtfer* ne peut placer l'entrée qu'à la position demandée car le HRTF est chargé à l'initialisation (souvenez-vous qu'actuellement Csound limite à 20 le nombre de fichiers qu'il peut garder en mémoire sans causer d'erreur de segmentation). Il faut ajuster la sortie soit en utilisant *balance* soit en la multipliant par une constante de mise à l'échelle.



### Note

Le taux d'échantillonnage de l'orchestre doit être de 44.1 kHz. C'est le taux auquel les HRTFs ont été mesurés. Si l'on veut utiliser les HRTFs à un taux différent, il faut les rééchantillonner au taux désiré.

## Exemples

Voici un exemple de l'opcode *hrtfer*. Il utilise les fichiers *hrtfer.csd* [exemples/hrtfer.csd], *HRTFcompact* [exemples/HRTFcompact] et *beats.wav* [exemples/beats.wav].

### Exemple 238. Exemple de l'opcode *hrtfer*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hrtfer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1
kaz          linseg 0, p3, -360 ; move the sound in circle
kel          linseg -40, p3, 45 ; around the listener, changing
                                ; elevation as its turning

asrc          soundin "beats.wav"
aleft,aright hrtfer asrc, kaz, kel, "HRTFcompact"
aleftscale    = aleft * 200
arightscale    = aright * 200

outs          aleftscale, arightscale
endin

</CsInstruments>
<CsScore>

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*hrtfmove, hrtfmove2, hrtfstat.*

## Crédits

Auteurs : Eli Breder et David MacIntyre  
Montréal  
1996

Correction de l'exemple grâce à un message d'Istvan Varga.

# hrtfmove

**hrtfmove** — Generates dynamic 3d binaural audio for headphones using magnitude interpolation and phase truncation.

## Description

This opcode takes a source signal and spatialises it in the 3 dimensional space around a listener by convolving the source with stored head related transfer function (HRTF) based filters.

## Syntax

```
aleft, aright hrtfmove asrc, kAz, kElev, ifilel, ifiler [, imode, ifade, isr]
```

## Initialization

## Initialization

*ifilel* -- left HRTF spectral data file

*ifiler* -- right HRTF spectral data file



### Note

Spectral datafiles (based on the MIT HRTF database) are available in 3 different sampling rates: 44.1, 48 and 96 khz and are labelled accordingly. Input and processing *sr* should match datafile *sr*. Files should be in the current directory or the SADIR (see *Environment Variables*).

*imode* -- optional, default 0 for phase truncation, 1 for minimum phase

*ifade* -- optional, number of processing buffers for phase change crossfade (default 8). Legal range is 1-24. A low value is recommended for complex sources (4 or less: a higher value may make the crossfade audible), a higher value (8 or more: a lower value may make the inconsistency when the filter changes phase values audible) for narrowband sources. Does not effect minimum phase processing.



### Note

Ocasionally fades may overlap (when unnaturally fast/complex trajectories are requested). In this case, a warning will be printed. Use a smaller crossfade or slightly change trajectory to avoid any possible inconsistencies that may arise.

*isr* - optional, default 44.1kHz, legal values: 44100, 48000 and 96000.

*kAz* -- azimuth value in degrees. Positive values represent position on the right, negative values are positions on the left.

*kElev* -- elevation value in degrees. Positive values represent position above horizontal, negative values are positions below horizontal (min -40).

Artifact-free user-defined trajectories are made possible using an interpolation algorithm based on spec-

tral magnitude interpolation and phase truncation. Crossfades are implemented to minimise/eliminate any inconsistencies caused by updating phase values. These crossfades are performed over a user definable number of convolution processing buffers. Complex sources may only need to crossfade over 1 buffer; narrow band sources may need several. The opcode also offers minimum phase based processing, a more traditional and complex method. In this mode, the hrtf filters used are reduced to minimum phase representations and the interpolation process then uses the relationship between minimum phase magnitude and phase spectra. Interaural time difference, which is inherent to the phase truncation process, is reintroduced in the minimum phase process using variable delay lines.

## Examples

Here is an example of the hrtfmove opcode. It uses the file *hrtfmove.csd* [examples/hrtfmove.csd].

### Exemple 239. Example of the hrtfmove opcode.

```
<CsoundSynthesizer>
<CsOptions>
; realtime audio out
; -o dac
; For Non-realtime output leave only the line below:
-o hrtf.wav
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 2

gasrc init 0

instr 1          ;a plucked string

    kamp = p4
    kcps = cpspch(p5)
    icps = cpspch(p5)

    al pluck kamp, kcps, icps, 0, 1

    gasrc = al

endin

instr 10 ;uses output from instr1 as source

    kaz linseg 0, p3, 720          ;2 full rotations

    ; Make sure you adjust these paths for your system
    aleft,aright hrtfmove gasrc, kaz,0, \
        "C:\Program Files\csound\samples\hrtf-44100-left.dat", \
        "C:\Program Files\csound\samples\hrtf-44100-right.dat"

    outs aleft, aright

endin

</CsInstruments>
<CsScore>

; Play Instrument 1: a simple arpeggio
i1 0 .2 15000 8.00
i1 + .2 15000 8.04
i1 + .2 15000 8.07
i1 + .2 15000 8.11
i1 + .2 15000 9.02
i1 + 1.5 15000 8.11
i1 + 1.5 15000 8.07
i1 + 1.5 15000 8.04
i1 + 1.5 15000 8.00
i1 + 1.5 15000 7.09
i1 + 1.5 15000 8.00
```

```
; Play Instrument 10 for 10 seconds.  
i10 0 10  
  
</CsScore>  
</CsoundSynthesizer>
```

## See Also

*hrtfmove2, hrtfstat, hrtfer.*

## Credits

Author: Brian Carty  
Maynooth  
2008

# hrtfmove2

**hrtfmove2** — Generates dynamic 3d binaural audio for headphones using a Woodworth based spherical head model with improved low frequency phase accuracy.

## Description

This opcode takes a source signal and spatialises it in the 3 dimensional space around a listener using head related transfer function (HRTF) based filters.

## Syntax

```
aleft, aright hrtfmove2 asrc, kAz, kElev, ifilel, ifiler [,ioverlap, iradius, isr]
```

## Initialization

*ifilel* -- left HRTF spectral data file

*ifiler* -- right HRTF spectral data file



### Note

Spectral datafiles (based on the MIT HRTF database) are available in 3 different sampling rates: 44.1, 48 and 96 khz and are labelled accordingly. Input and processing *sr* should match datafile *sr*. Files should be in the current directory or the SADIR (see *Environment Variables*).

*ioverlap* -- optional, number of overlaps for STFT processing (default 4). See STFT section of manual.

*iradius* -- optional, head radius used for phase spectra calculation in centimeters (default 9.0)

*isr* - optional, default 44.1kHz, legal values: 44100, 48000 and 96000.

## Performance

*asrc* -- Input/source signal.

*kAz* -- azimuth value in degrees. Positive values represent position on the right, negative values are positions on the left.

*kElev* -- elevation value in degrees. Positive values represent position above horizontal, negative values are positions below horizontal (min -40).

Artifact-free user-defined trajectories are made possible using an interpolation algorithm based on spectral magnitude interpolation and a derived phase spectrum based on the Woodworth spherical head model. Accuracy is increased for the data set provided by extracting and applying a frequency dependent scaling factor to the phase spectra, leading to a more precise low frequency interaural time difference. Users can control head radius for the phase derivation, allowing a crude level of individualisation. The dynamic source version of the opcode uses a Short Time Fourier Transform algorithm to avoid artefacts caused by derived phase spectra changes. STFT processing means this opcode is more computationally intensive than *hrtfmove* using phase truncation, but phase is constantly updated by *hrtfmove2*.



## Examples

Here is an example of the `hrtfmove2` opcode. It uses the file `hrtfmove2.csd` [examples/hrtfmove2.csd].

### Exemple 240. Example of the `hrtfmove2` opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select flags here
; realtime audio out
; -o dac
; For Non-realtime ouput leave only the line below:
-o hrtf.wav
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

gasrc init 0

instr 1          ;a plucked string

    kamp = p4
    kcps = cpspch(p5)
    icps = cpspch(p5)

    al pluck kamp, kcps, icps, 0, 1

    gasrc = al

endin

instr 10 ;uses output from instr1 as source

    kaz linseg 0, p3, 720          ;2 full rotations

    aleft,aright hrtfmove2 gasrc, kaz,0, "hrtf-44100-left.dat","hrtf-44100-right.dat"

    outs aleft, aright

endin

</CsInstruments>
<CsScore>

; Play Instrument 1: a simple arpeggio
i1 0 .2 15000 8.00
i1 + .2 15000 8.04
i1 + .2 15000 8.07
i1 + .2 15000 8.11
i1 + .2 15000 9.02
i1 + 1.5 15000 8.11
i1 + 1.5 15000 8.07
i1 + 1.5 15000 8.04
i1 + 1.5 15000 8.00
i1 + 1.5 15000 7.09
i1 + 1.5 15000 8.00

; Play Instrument 10 for 10 seconds.
i10 0 10

</CsScore>
</CsoundSynthesizer>
```

## See Also

*hrtfmove, hrtfstat, hrtfer.*

## Credits

Author: Brian Carty  
Maynooth  
2008

# hrtfstat

**hrtfstat** — Generates static 3d binaural audio for headphones using a Woodworth based spherical head model with improved low frequency phase accuracy.

## Description

This opcode takes a source signal and spatialises it in the 3 dimensional space around a listener using head related transfer function (HRTF) based filters. It produces a static output (azimuth and elevation parameters are i-rate), because a static source allows much more efficient processing than *hrtfmove* and *hrtfmove2*.

## Syntax

```
aleft, aright hrtfstat asrc, iAz, iElev, ifilel, ifiler [,iradius, isr]
```

## Initialization

*iAz* -- azimuth value in degrees. Positive values represent position on the right, negative values are positions on the left.

*iElev* -- elevation value in degrees. Positive values represent position above horizontal, negative values are positions below horizontal (min -40).

*ifilel* -- left HRTF spectral data file

*ifiler* -- right HRTF spectral data file



### Note

Spectral datafiles (based on the MIT HRTF database) are available in 3 different sampling rates: 44.1, 48 and 96 khz and are labelled accordingly. Input and processing sr should match datafile sr. Files should be in the current directory or the SADIR (see *Environment Variables*).

*iradius* -- optional, head radius used for phase spectra calculation in centimeters (default 9.0)

*isr* - optional (default 44.1kHz). Legal values are 44100, 48000 and 96000.

## Performance

Artifact-free user-defined static spatialisation is made possible using an interpolation algorithm based on spectral magnitude interpolation and a derived phase based on the Woodworth spherical head model. Accuracy is increased for the data set provided by extracting and applying a frequency dependent scaling factor to the phase spectra, leading to a more precise low frequency interaural time difference. Users can control head radius for the phase derivation, allowing a crude level of individualisation. The static source version of the opcode uses overlap add convolution (it does not need STFT processing, see *hrtfmove2*), and is thus considerably more efficient than *hrtfmove2* or *hrtfmove*, but cannot generate moving sources.

## Examples

It can be found in the file *htrfstat.csd* [examples/htrfstat.csd].

### Exemple 241. Example of the htrfstat opcode.

```
<CsoundSynthesizer>
<CsOptions>
  ; Select flags here
  ; realtime audio out
  ; -o dac
  ; For Non-realtime output leave only the line below:
  -o hrtf.wav

</CsOptions>
<CsInstruments>

  sr = 44100
  kr = 4410
  ksmps = 10
  nchnls = 2

  gasrc init 0

  instr 1          ;a plucked string

  kamp = p4
  kcps = cpspch(p5)
  icps = cpspch(p5)

  a1 pluck kamp, kcps, icps, 0, 1

  gasrc = a1

  endin

  instr 10 ;uses output from instr1 as source

  aleft,aright htrfstat gasrc, 90,0, "hrtf-44100-left.dat","hrtf-44100-right.dat"

  outs aleft, aright

  endin

</CsInstruments>
<CsScore>

  ; Play Instrument 1: a plucked string
  i1 0 2 20000 8.00

  ; Play Instrument 10 for 2 seconds.
  i10 0 2

</CsScore>
</CsoundSynthesizer>
```

## See Also

*hrtfmove*, *hrtfmove2*, *hrtfer*.

## Credits

Author: Brian Carty  
Maynooth

2008

# hsboscil

hsboscil — Un oscillateur qui prend en arguments l'intonation et la brillance.

## Description

Un oscillateur qui prend en arguments l'intonation et la brillance, relativement à une fréquence de base.

## Syntaxe

```
ares hsboscil kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn \  
    [, ioctcnt] [, iphs]
```

## Initialisation

*ibasfreq* -- fréquence de base par rapport à laquelle l'intonation et la brillance sont relatives.

*iwfn* -- table de fonction de la forme d'onde, habituellement une sinus.

*ioctfn* -- table de fonction utilisée pour pondérer les octaves, habituellement quelque chose comme

**f**1 0 1024 -19 1 0.5 270 0.5

*ioctcnt* (facultatif) -- nombre d'octaves utilisées pour le mélange de brillance. Doit valoir entre 2 et 10. Par défaut = 3.

*iphs* (facultatif, par défaut = 0) -- phase initiale de l'oscillateur. Si *iphs* = -1, l'initialisation est ignorée.

## Exécution

*kamp* -- amplitude de la note

*ktone* -- paramètre cyclique d'intonation cyclique relatif à *ibasfreq* en octave logarithmique, entre 0 et 1, des valeurs > 1 peuvent être utilisées, et sont réduites en interne à *frac(ktone)*.

*kbrite* -- paramètre de brillance relatif à *ibasfreq*, obtenue en pondérant *ioctcnt* octaves. Il est échelonné de telle manière qu'une valeur de 0 correspond à la valeur originale de *ibasfreq*, 1 correspond à une octave au-dessus de *ibasfreq*, -2 correspond à deux octaves sous *ibasfreq*, etc. *kbrite* peut être fractionnaire.

*hsboscil* prend en arguments l'intonation et la brillance, relativement à une fréquence de base (*ibasfreq*). L'intonation est un paramètre cyclique dans l'octave logarithmique, la brillance est réalisée en mélangeant plusieurs octaves pondérées. Il est utile lorsque l'espace d'intonation est appréhendé dans un concept de coordonnées polaires.

Si *ktone* est une droite et *kbrite* une constante, le résultat produit est le glissando de Risset.

La table de l'oscillateur *iwfn* est toujours lue avec interpolation. Le temps d'exécution est approximativement *ioctcnt* \* *oscili*.

## Exemples

Voici un exemple de l'opcode `hsboscil`. Il utilise le fichier `hsboscil.csd` [examples/hsboscil.csd].

### Exemple 242. Exemple de l'opcode `hsboscil`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hsboscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; synth waveform
giwave ftgen 1, 0, 1024, 10, 1, 1, 1, 1
; blending window
giblend ftgen 2, 0, 1024, -19, 1, 0.5, 270, 0.5

; Instrument #1 - produces Risset's glissando.
instr 1
  kamp = 10000
  kbrite = 0.5
  ibasfreq = 200
  ioctcnt = 5

  ; Change ktone linearly from 0 to 1,
  ; over the period defined by p3.
  ktone line 0, p3, 1

  al hsboscil kamp, ktone, kbrite, ibasfreq, giwave, giblend, ioctcnt
  out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode `hsboscil` dans un instrument MIDI. Il utilise le fichier `hsboscil_midi.csd` [examples/hsboscil\_midi.csd].

### Exemple 243. Exemple de l'opcode `hsboscil` dans un instrument MIDI.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in   No messages  MIDI in
-odac        -iadc       -d           -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o hsboscil_midi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; synth waveform
giwave ftgen 1, 0, 1024, 10, 1, 1, 1, 1
; blending window
giblend ftgen 2, 0, 1024, -19, 1, 0.5, 270, 0.5

; Instrument #1 - use hsboscil in a MIDI instrument.
instr 1
  ibase = cpsoct(6)
  ioctcnt = 5

  ; all octaves sound alike.
  itona octmidi
  ; velocity is mapped to brightness
  ibrite ampmidi 3

  ; Map an exponential envelope for the amplitude.
  kenv expon 20000, 1, 100

  asig hsboscil kenv, itona, ibrite, ibase, giwave, giblend, ioctcnt
  out asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten minutes
i 1 0 600
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Peter Neubäcker  
 Munich, Allemagne  
 Août 1999

Nouveau dans la version 3.58 de Csound



# hvs1

**hvs1** — Allows one-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

## Description

*hvs1* allows one-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

## Syntax

```
hvs1 kx, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]
```

## Initialization

*inumParms* - number of parameters controlled by the HVS. Each HVS snapshot is made up of *inumParms* elements.

*inumPointsX* - number of points that each dimension of the HVS cube (or square in case of two-dimensional HVS; or line in case of one-dimensional HVS) is made up.

*iOutTab* - number of the table receiving the set of output-parameter instant values of the HVS. The total amount of parameters is defined by the *inumParms* argument.

*iPositionsTab* – a table filled with the individual positions of snapshots in the HVS matrix (see below for more information).

*iSnapTab* – a table filled with all the snapshots. Each snapshot is made up of a set of parameter values. The amount of elements contained in each snapshots is specified by the *inumParms* argument. The set of elements of each snapshot follows (and is adjacent) to the previous one in this table. So the total size of this table should be  $\geq$  to *inumParms* multiplied the number of snapshots you intend to store for the HVS.

*iConfigTab* – (optional) a table containing the behavior of the HVS for each parameter. If the value of *iConfigTab* is zero (default), this argument is ignored, meaning that each parameter is treated with linear interpolation by the HVS. If *iConfigTab* is different than zero, then it must refer to an existing table whose contents are in its turn referring to a particular kind of interpolation. In this table, a value of -1 indicates that corresponding parameter is leaved unchanged (ignored) by the HVS; a value of zero indicates that corresponding parameter is treated with linear-interpolation; each other values must be integer numbers indicating an existing table filled with a shape which will determine the kind of special interpolation to be used (table-based interpolation).

## Performance

*kx* - these are externally-modified variables which controls the motion of the pointer in the HVS matrix cube (or square or line in case of HVS matrices made up of less than 3 dimensions). The range of these input arguments must be 0 to 1.

Hyper Vectorial Synthesis is a technique that allows control of a huge set of parameters by using a simple and global approach. The key concepts of the HVS are:

The set of HVS parameters, whose amount is fixed and defined by the *inumParms* argument. During the

HVS performance, all these parameters are variant and can be applied to any sound synthesis technique, as well as to any global control for algorithmic composition and any other kind of level. The user must previously define several sets of fixed values for each HVS parameter, each set corresponding to a determinate synthesis configuration. Each set of values is called snapshot, and can be considered as the coordinates of a bound of a multi-dimensional space. The HVS consists on moving a point in this multi-dimensional space (by using a special motion pointer, see below), according and inside the bounds defined by the snapshots. You can fix any amount of HVS parameters (each parameter being a dimension of the multi-dimensional space), even a huge number, the limit only depends on the processing power (and the memory) of your computer and on the complexity of the sound-synthesis you will use.

The HVS cube (or square or line). This is the matrix (of 3, 2 or 1 dimensions, according to the hvs opcode you intend to use) of “mainstays” (or pivot) points of HVS. The total amount of pivot-points depends on the value of the *inumPointsX*, *inumPointsY* and *inumPointsZ* arguments. In the case of a 3-dimensional HVS matrix you can define, for instance, 3 points for the X dimension, 5 for the Y dimension and 2 for the Z dimension. In this case, the total number of pivot-points is  $3 * 5 * 2 = 30$ . With this set of pivot points, the cube is divided into smaller cubed zones each one bounded by eight nearby points. Each point is numbered. The numeral order of these points is established in the following way: number zero is the first point, number 1 the second and so on. Assuming you are using a 3-dimensional HVS cube having the number of points above mentioned (i.e. 3, 5 and 2 respectively for the X, Y and Z axis), the first point (point zero) is the upper-left-front vertex of the cube, by facing the XY plane of the cube. The second point is the middle point of the upper front edge of the cube and so on. You can refer to the figure below in order to understand how the numeral order of the pivot-points proceeds:

For the 2-dimensional HVS, it is the same, by only omitting the rear cube face, so each zone is bounded by 4 pivot-points instead of 8. For the 1-dimensional HVS, the whole thing is even simpler because it is a line with the pivot-points proceeding from left to right. Each point is coupled with a snapshot.

Snapshot order, as stored into the *iSnapTab*, can or cannot follow the order of the pivot-points numbers. In fact it is possible to alter this order by means the *iPositionsTab*, a table that remaps the position of each snapshot in relation to the pivot points. The *iPositionsTab* is made up of the positions of the snapshots (contained in the *iSnapTab*) in the two-dimensional grid. Each subsequent element is actually a pointer representing the position in the *iSnapTab*. For example, in a 2-dimensional HVS matrix such as the following (in this case having *inumPointsX* = 3 and *inumPointsY* = 5:

**Tableau 10.**

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

These numbers (to be stored in the *iSnapTab* table by using, for instance, the GEN02 function generator) represents the snapshot position within the grid (in this case a 3x5 matrix). So, the first element 5, has index zero and represents the 6th (element zero is the first) snapshot contained in the *iSnapTab*, the second element 7 represents the 8th element of *iSnapTab* and so on. Summing up, the vertices of each zone (a cubed zone is delimited by 8 vertices; a squared zone by 4 vertices and a linear zone by 2 points) are coupled with a determinate snapshot, whose number is remapped by the *iSnapTab*.

Output values of the HVS are influenced by the motion pointer, a point whose position, in the HVS cube (or square or segment) is determined by the *kx*, *ky* and *kz* arguments. The values of these arguments, which must be in the 0 to 1 range, are externally set by the user. The output values, whose amount is equal to the *inumParms* argument, are stored in the *iOutTab*, a table that must be already allocated by the user, and must be at least *inumParms* size. In what way the motion pointer influences the output? Well, when the motion pointer falls in a determinate cubed zone, delimited, for instance, by 8 vertices

(or pivot points), we assume that each vertex has associated a different snapshot (i.e. a set of *inumParms* values), well, the output will be the weighted average value of the 8 vertices, calculated according on the distance of the motion pointer from each of the 8 vertices. In the case of a default behavior, when the *iConfigTab* argument is not set, the exact output is calculated by using linear interpolation which is applied to each different parameter of the HVS. Anyway, it is possible to influence this behavior by setting the *iConfigTab* argument to a number of a table whose contents can affect one or more HVS parameters. The *iConfigTab* table elements are associated to each HVS parameter and their values affect the HVS output in the following way:

- If *iConfigTab* is equal to -1, corresponding output is skipped, i.e. the element is not calculated, leaving corresponding element value in the *iOutTab* unchanged;
- If *iConfigTab* is equal to zero, then the normal HVS output is calculated (by using weighted average of the nearest vertex of current zone where it falls the motion pointer);
- If *iConfigTab* element is equal to an integer number > zero, then the contents of a table having that number is used as a shape of a table-based interpolation.

## Examples

Here is an example of the hvs1 opcode. It uses the file *hvs1.csd* [examples/hvs1.csd].

### Exemple 244. Example of the hvs1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac             -iadac       -d             ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=100
nchnls=2
0dbfs = 1

; Example by Gabriel Maldonado and Andres Cabrera

ginumLinesX init 16
ginumParms  init 3

giOutTab ftgen 5,0,8, -2,      0
giPosTab ftgen 6,0,32, -2,    3,2,1,0,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
giSnapTab ftgen 8,0,64, -2,    1,1,1,  2,0,0,  3,2,0,  2,2,2,  5,2,1,  2,3,4,  6,1,7,  0,0,0, \
                                1,3,5,   3,4,4,  1,5,8,  1,1,5,  4,3,2,  3,4,5,  7,6,5,  7,8,9
tb0_init giOutTab

FLpanel "hsv1",500,100,10,10,0
gk1,ih1 FLslider "X", 0,1, 0,5, -1, 400,30, 50,20
FLpanel_end
FLrun

instr 1
;      kx,      inumParms, inumPointsX, iOutTab, iPosTab, iSnapTab [, iConfigTab]
hvs1      gk1,      ginumParms, ginumLinesX, giOutTab, giPosTab, giSnapTab ;; iConfigTab

k0 init 0
k1 init 1
k2 init 2
```

```

printk2 tb0(k0)
printk2 tb0(k1), 10
printk2 tb0(k2), 20

aosc1 oscil tb0(k0)/20, tb0(k1)*100 + 200, 1
aosc2 oscil tb0(k1)/20, tb0(k2)*100 + 200, 1
aosc3 oscil tb0(k2)/20, tb0(k0)*100 + 200, 1
aosc4 oscil tb0(k1)/20, tb0(k0)*100 + 200, 1
aosc5 oscil tb0(k2)/20, tb0(k1)*100 + 200, 1
aosc6 oscil tb0(k0)/20, tb0(k2)*100 + 200, 1

outs aosc1 + aosc2 + aosc3, aosc4 + aosc5 + aosc6
    endin

</CsInstruments>
<CsScore>

f1 0 1024 10 1
f0 3600
i1 0 3600

</CsScore>
</CsoundSynthesizer>

```

## See Also

*hvs2, hvs3, vphaseseg*

## Credits

Author: Gabriel Maldonado

New in version 5.06

## hvs2

**hvs2** — Allows two-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

## Description

*hvs2* allows two-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

## Syntax

```
hvs2 kx, ky, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]
```

## Initialization

*inumParms* - number of parameters controlled by the HVS. Each HVS snapshot is made up of *inumParms* elements.

*inumPointsX* - number of points that each dimension of the HVS cube (or square in case of two-dimensional HVS; or line in case of one-dimensional HVS) is made up.

*iOutTab* - number of the table receiving the set of output-parameter instant values of the HVS. The total amount of parameters is defined by the *inumParms* argument.

*iPositionsTab* – a table filled with the individual positions of snapshots in the HVS matrix (see below for more information).

*iSnapTab* – a table filled with all the snapshots. Each snapshot is made up of a set of parameter values. The amount of elements contained in each snapshots is specified by the *inumParms* argument. The set of elements of each snapshot follows (and is adjacent) to the previous one in this table. So the total size of this table should be  $\geq$  to *inumParms* multiplied the number of snapshots you intend to store for the HVS.

*iConfigTab* – (optional) a table containing the behavior of the HVS for each parameter. If the value of *iConfigTab* is zero (default), this argument is ignored, meaning that each parameter is treated with linear interpolation by the HVS. If *iConfigTab* is different than zero, then it must refer to an existing table whose contents are in its turn referring to a particular kind of interpolation. In this table, a value of -1 indicates that corresponding parameter is leaved unchanged (ignored) by the HVS; a value of zero indicates that corresponding parameter is treated with linear-interpolation; each other values must be integer numbers indicating an existing table filled with a shape which will determine the kind of special interpolation to be used (table-based interpolation).

## Performance

*kx*, *ky* - these are externally-modified variables which controls the motion of the pointer in the HVS matrix cube (or square or line in case of HVS matrices made up of less than 3 dimensions). The range of these input arguments must be 0 to 1.

Hyper Vectorial Synthesis is a technique that allows control of a huge set of parameters by using a simple and global approach. The key concepts of the HVS are:

The set of HVS parameters, whose amount is fixed and defined by the *inumParms* argument. During the

HVS performance, all these parameters are variant and can be applied to any sound synthesis technique, as well as to any global control for algorithmic composition and any other kind of level. The user must previously define several sets of fixed values for each HVS parameter, each set corresponding to a determinate synthesis configuration. Each set of values is called snapshot, and can be considered as the coordinates of a bound of a multi-dimensional space. The HVS consists on moving a point in this multi-dimensional space (by using a special motion pointer, see below), according and inside the bounds defined by the snapshots. You can fix any amount of HVS parameters (each parameter being a dimension of the multi-dimensional space), even a huge number, the limit only depends on the processing power (and the memory) of your computer and on the complexity of the sound-synthesis you will use.

The HVS cube (or square or line). This is the matrix (of 3, 2 or 1 dimensions, according to the hvs opcode you intend to use) of “mainstays” (or pivot) points of HVS. The total amount of pivot-points depends on the value of the *inumPointsX*, *inumPointsY* and *inumPointsZ* arguments. In the case of a 3-dimensional HVS matrix you can define, for instance, 3 points for the X dimension, 5 for the Y dimension and 2 for the Z dimension. In this case, the total number of pivot-points is  $3 * 5 * 2 = 30$ . With this set of pivot points, the cube is divided into smaller cubed zones each one bounded by eight nearby points. Each point is numbered. The numeral order of these points is established in the following way: number zero is the first point, number 1 the second and so on. Assuming you are using a 3-dimensional HVS cube having the number of points above mentioned (i.e. 3, 5 and 2 respectively for the X, Y and Z axis), the first point (point zero) is the upper-left-front vertex of the cube, by facing the XY plane of the cube. The second point is the middle point of the upper front edge of the cube and so on. You can refer to the figure below in order to understand how the numeral order of the pivot-points proceeds:

For the 2-dimensional HVS, it is the same, by only omitting the rear cube face, so each zone is bounded by 4 pivot-points instead of 8. For the 1-dimensional HVS, the whole thing is even simpler because it is a line with the pivot-points proceeding from left to right. Each point is coupled with a snapshot.

Snapshot order, as stored into the *iSnapTab*, can or cannot follow the order of the pivot-points numbers. In fact it is possible to alter this order by means the *iPositionsTab*, a table that remaps the position of each snapshot in relation to the pivot points. The *iPositionsTab* is made up of the positions of the snapshots (contained in the *iSnapTab*) in the two-dimensional grid. Each subsequent element is actually a pointer representing the position in the *iSnapTab*. For example, in a 2-dimensional HVS matrix such as the following (in this case having *inumPointsX* = 3 and *inumPointsY* = 5:

**Tableau 11.**

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

These numbers (to be stored in the *iSnapTab* table by using, for instance, the GEN02 function generator) represents the snapshot position within the grid (in this case a 3x5 matrix). So, the first element 5, has index zero and represents the 6th (element zero is the first) snapshot contained in the *iSnapTab*, the second element 7 represents the 8th element of *iSnapTab* and so on. Summing up, the vertices of each zone (a cubed zone is delimited by 8 vertices; a squared zone by 4 vertices and a linear zone by 2 points) are coupled with a determinate snapshot, whose number is remapped by the *iSnapTab*.

Output values of the HVS are influenced by the motion pointer, a point whose position, in the HVS cube (or square or segment) is determined by the *kx*, *ky* and *kz* arguments. The values of these arguments, which must be in the 0 to 1 range, are externally set by the user. The output values, whose amount is equal to the *inumParms* argument, are stored in the *iOutTab*, a table that must be already allocated by the user, and must be at least *inumParms* size. In what way the motion pointer influences the output? Well, when the motion pointer falls in a determinate cubed zone, delimited, for instance, by 8 vertices

(or pivot points), we assume that each vertex has associated a different snapshot (i.e. a set of *inumParms* values), well, the output will be the weighted average value of the 8 vertices, calculated according on the distance of the motion pointer from each of the 8 vertices. In the case of a default behavior, when the *iConfigTab* argument is not set, the exact output is calculated by using linear interpolation which is applied to each different parameter of the HVS. Anyway, it is possible to influence this behavior by setting the *iConfigTab* argument to a number of a table whose contents can affect one or more HVS parameters. The *iConfigTab* table elements are associated to each HVS parameter and their values affect the HVS output in the following way:

- If *iConfigTab* is equal to -1, corresponding output is skipped, i.e. the element is not calculated, leaving corresponding element value in the *iOutTab* unchanged;
- If *iConfigTab* is equal to zero, then the normal HVS output is calculated (by using weighted average of the nearest vertex of current zone where it falls the motion pointer);
- If *iConfigTab* element is equal to an integer number > zero, then the contents of a table having that number is used as a shape of a table-based interpolation.

## Examples

Here is an example of the hvs2 opcode. It uses the file *hvs2.csd* [examples/hvs2.csd].

### Exemple 245. Example of the hvs2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d            ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=100
nchnls=2

0dbfs = 1

ginumLinesX init 4
ginumLinesY init 4
ginumParms init 3

giOutTab ftgen 5,0,8, -2,      0
giPosTab ftgen 6,0,32, -2,    3,2,1,0,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
giSnapTab ftgen 8,0,64, -2,   1,1,1,  2,0,0,  3,2,0,  2,2,2,  5,2,1,  2,3,4,  6,1,7,  0,0,0, \
                             1,3,5,   3,4,4,  1,5,8,  1,1,5,  4,3,2,  3,4,5,  7,6,5,  7,8,9

tb0_init giOutTab

      FLpanel "Prova HVS2",600,400,10,100,0

gk1,   gk2,   ih1, ih2  FLjoy  "HVS controller XY", 0,    1,    1,    0,    0,    0,    -1,
; *ihandle,
gihandle FLhvsBox ginumLinesX,  *numlinesX,  *numlinesY, *iwidth, *iheight, *ix, *iy,*image;
                             ginumLinesY,  300,   300,   300,  50,  1

      FLpanel_end
      FLrun

instr 1
```

```

; Smooth control signals to avoid clicks
kx portk gk1, 0.02
ky portk gk2, 0.02

;          kx, ky, inumParms, inumlinesX, inumlinesY, iOutTab, iPosTab, iSnapTab [, iConfigT
hvs2 kx, ky, ginumParms, ginumLinesX, ginumLinesY, giOutTab, giPosTab, giSnapTab ;, iConfigT

;          *kx, *ky, *ihandle;
FLhvsBoxSetValue gk1, gk2, gihandle

k0 init 0
k1 init 1
k2 init 2

printk2 tb0(k0)
printk2 tb0(k1), 10
printk2 tb0(k2), 20

kris init 0.003
kdur init 0.02
kdec init 0.007

; Make parameters of synthesis depend on the table values produced by hvs
ares1 fof 0.2, tb0(k0)*100 + 50, tb0(k1)*100 + 200, 0, tb0(k2) * 10 + 50, 0.003, 0.02, 0.007, 20, \
1, 2, p3
ares2 fof 0.2, tb0(k1)*100 + 50, tb0(k2)*100 + 200, 0, tb0(k0) * 10 + 50, 0.003, 0.02, 0.007, 20, \
1, 2, p3

outs ares1, ares2
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1 ;Sine wave
f 2 0 1024 19 0.5 0.5 270 0.5 ;Grain envelope table

f0 3600

i1 0 3600

</CsScore>
</CsSoundSynthesizer>

```

Here is second example of the hvs2 opcode. It uses the file *hvs2.csd* [examples/hvs2-2.csd].

## Exemple 246. Example of the hvs2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in
-odac -iadc ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hvs2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=48000
ksmps=100
nchnls=2

; Example by James Hearon 2008
; Edited by Andres Cabrera

ginumPointsX init 16
ginumPointsY init 16
ginumParms init 3

;Generate 9 tables with arbitrary points
gitmp ftgen 100, 0, 16, -2, 70, 260, 390, 180, 200, 300, 980, 126, \

```



```

        330, 860, 580, 467, 220, 399, 1026, 1500
gitmp ftgen 200, 0, 16, -2, 100, 200, 300, 140, 600, 700, 880, 126, \
        330, 560, 780, 167, 220, 999, 1026, 1500
gitmp ftgen 300, 0, 16, -2, 400, 200, 300, 540, 600, 700, 880, 126, \
        330, 160, 780, 167, 820, 999, 1026, 1500
gitmp ftgen 400, 0, 16, -2, 100, 200, 800, 640, 600, 300, 880, 126, \
        330, 660, 780, 167, 220, 999, 1026, 1500
gitmp ftgen 500, 0, 16, -2, 200, 200, 360, 440, 600, 700, 880, 126, \
        330, 560, 380, 167, 220, 499, 1026, 1500
gitmp ftgen 600, 0, 16, -2, 100, 600, 300, 840, 600, 700, 880, 126, \
        330, 260, 980, 367, 120, 399, 1026, 1500
gitmp ftgen 700, 0, 16, -2, 100, 200, 300, 340, 200, 500, 380, 126, \
        330, 860, 780, 867, 120, 999, 1026, 1500
gitmp ftgen 800, 0, 16, -2, 100, 600, 300, 240, 200, 700, 880, 126, \
        130, 560, 980, 167, 220, 499, 1026, 1500
gitmp ftgen 900, 0, 16, -2, 100, 800, 200, 140, 600, 700, 680, 126, \
        330, 560, 780, 167, 120, 299, 1026, 1500

giOutTab ftgen 5,0,8, -2, 0
giPosTab ftgen 6,0,32, -2, 0,1,2,3,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
giSnapTab ftgen 8,0,64, -2, 1,1,1, 2,0,0, 3,2,0, 2,2,2, \
        5,2,1, 2,3,4, 6,1,7, 0,0,0, 1,3,5, 3,4,4, 1,5,8, 1,1,5, \
        4,3,2, 3,4,5, 7,6,5, 7,8,9

tb0_init giOutTab

        FLpanel "hsv2",440,100,10,10,0
gk1,ih1 FLslider "X", 0,1, 0, 5, -1, 400,20, 20,10
gk2, ih2 FLslider "Y", 0, 1, 0, 5, -1, 400, 20, 20, 50
        FLpanel_end

        FLpanel "hvsBox",280,280,500,1000,0
;ihandle FLhvsBox inumlinesX, inumlinesY, iwidth, iheight, ix, iy [, image]
gihl FLhvsBox 16, 16, 250, 250, 10, 1
        FLpanel_end
FLrun

        instr 1
FLhvsBoxSetValue gk1, gk2, gihl

hvs2 gk1,gk2, ginumParms, ginumPointsX, ginumPointsY, giOutTab, giPosTab, giSnapTab ;, iConfigTab

k0 init 0
k1 init 1
k2 init 2
kspeed init 0

kspeed = int((tb0(k2)) + 1)*.10

kenv oscil 25000, kspeed*16, 10

k1 phasor kspeed ;slow phasor: 200 sec.
kpch tableikt k1 * 16, int((tb0(k1)) +1)*100 ;scale phasor * length
a1 oscilikt kenv, kpch, int(tb0(k0)) +1000;scale pitch slightly
ahp butterlp a1, 2500
outs ahp, ahp

        endin

</CsInstruments>
<CsScore>

f 10 0 1024 20 5 ;use of windowing function
f1000 0 1024 10 .33 .25 .5
f1001 0 1024 10 1
f1002 0 1024 10 .5 .25 .05
f1003 0 1024 10 .05 .10 .3 .5 1
f1004 0 1024 10 1 .5 .25 .125 .625
f1005 0 1024 10 .33 .44 .55 .66
f1006 0 1024 10 1 1 1 1 1
f1007 0 1024 10 .05 .25 .05 .25 .05 1

f0 3600
i1 0 3600

</CsScore>
</CsoundSynthesizer>

```

## See Also

*hvs1, hvs3, vphaseseg*

## Credits

Author: Gabriel Maldonado

New in version 5.06

## hvs3

**hvs3** — Allows three-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

## Description

*hvs3* allows three-dimensional Hyper Vectorial Synthesis (HVS) controlled by externally-updated k-variables.

## Syntax

```
hvs3 kx, ky, kz, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]
```

## Initialization

*inumParms* - number of parameters controlled by the HVS. Each HVS snapshot is made up of *inumParms* elements.

*inumPointsX* - number of points that each dimension of the HVS cube (or square in case of two-dimensional HVS; or line in case of one-dimensional HVS) is made up.

*iOutTab* - number of the table receiving the set of output-parameter instant values of the HVS. The total amount of parameters is defined by the *inumParms* argument.

*iPositionsTab* – a table filled with the individual positions of snapshots in the HVS matrix (see below for more information).

*iSnapTab* – a table filled with all the snapshots. Each snapshot is made up of a set of parameter values. The amount of elements contained in each snapshots is specified by the *inumParms* argument. The set of elements of each snapshot follows (and is adjacent) to the previous one in this table. So the total size of this table should be  $\geq$  to *inumParms* multiplied the number of snapshots you intend to store for the HVS.

*iConfigTab* – (optional) a table containing the behavior of the HVS for each parameter. If the value of *iConfigTab* is zero (default), this argument is ignored, meaning that each parameter is treated with linear interpolation by the HVS. If *iConfigTab* is different than zero, then it must refer to an existing table whose contents are in its turn referring to a particular kind of interpolation. In this table, a value of -1 indicates that corresponding parameter is leaved unchanged (ignored) by the HVS; a value of zero indicates that corresponding parameter is treated with linear-interpolation; each other values must be integer numbers indicating an existing table filled with a shape which will determine the kind of special interpolation to be used (table-based interpolation).

## Performance

*kx*, *ky*, *kz* - these are externally-modified variables which controls the motion of the pointer in the HVS matrix cube (or square or line in case of HVS matrices made up of less than 3 dimensions). The range of these input arguments must be 0 to 1.

Hyper Vectorial Synthesis is a technique that allows control of a huge set of parameters by using a simple and global approach. The key concepts of the HVS are:

The set of HVS parameters, whose amount is fixed and defined by the *inumParms* argument. During the

HVS performance, all these parameters are variant and can be applied to any sound synthesis technique, as well as to any global control for algorithmic composition and any other kind of level. The user must previously define several sets of fixed values for each HVS parameter, each set corresponding to a determinate synthesis configuration. Each set of values is called snapshot, and can be considered as the coordinates of a bound of a multi-dimensional space. The HVS consists on moving a point in this multi-dimensional space (by using a special motion pointer, see below), according and inside the bounds defined by the snapshots. You can fix any amount of HVS parameters (each parameter being a dimension of the multi-dimensional space), even a huge number, the limit only depends on the processing power (and the memory) of your computer and on the complexity of the sound-synthesis you will use.

The HVS cube (or square or line). This is the matrix (of 3, 2 or 1 dimensions, according to the hvs opcode you intend to use) of “mainstays” (or pivot) points of HVS. The total amount of pivot-points depends on the value of the *inumPointsX*, *inumPointsY* and *inumPointsZ* arguments. In the case of a 3-dimensional HVS matrix you can define, for instance, 3 points for the X dimension, 5 for the Y dimension and 2 for the Z dimension. In this case, the total number of pivot-points is  $3 * 5 * 2 = 30$ . With this set of pivot points, the cube is divided into smaller cubed zones each one bounded by eight nearby points. Each point is numbered. The numeral order of these points is established in the following way: number zero is the first point, number 1 the second and so on. Assuming you are using a 3-dimensional HVS cube having the number of points above mentioned (i.e. 3, 5 and 2 respectively for the X, Y and Z axis), the first point (point zero) is the upper-left-front vertex of the cube, by facing the XY plane of the cube. The second point is the middle point of the upper front edge of the cube and so on. You can refer to the figure below in order to understand how the numeral order of the pivot-points proceeds:

For the 2-dimensional HVS, it is the same, by only omitting the rear cube face, so each zone is bounded by 4 pivot-points instead of 8. For the 1-dimensional HVS, the whole thing is even simpler because it is a line with the pivot-points proceeding from left to right. Each point is coupled with a snapshot.

Snapshot order, as stored into the *iSnapTab*, can or cannot follow the order of the pivot-points numbers. In fact it is possible to alter this order by means the *iPositionsTab*, a table that remaps the position of each snapshot in relation to the pivot points. The *iPositionsTab* is made up of the positions of the snapshots (contained in the *iSnapTab*) in the two-dimensional grid. Each subsequent element is actually a pointer representing the position in the *iSnapTab*. For example, in a 2-dimensional HVS matrix such as the following (in this case having *inumPointsX* = 3 and *inumPointsY* = 5:

**Tableau 12.**

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

These numbers (to be stored in the *iSnapTab* table by using, for instance, the GEN02 function generator) represents the snapshot position within the grid (in this case a 3x5 matrix). So, the first element 5, has index zero and represents the 6th (element zero is the first) snapshot contained in the *iSnapTab*, the second element 7 represents the 8th element of *iSnapTab* and so on. Summing up, the vertices of each zone (a cubed zone is delimited by 8 vertices; a squared zone by 4 vertices and a linear zone by 2 points) are coupled with a determinate snapshot, whose number is remapped by the *iSnapTab*.

Output values of the HVS are influenced by the motion pointer, a point whose position, in the HVS cube (or square or segment) is determined by the *kx*, *ky* and *kz* arguments. The values of these arguments, which must be in the 0 to 1 range, are externally set by the user. The output values, whose amount is equal to the *inumParms* argument, are stored in the *iOutTab*, a table that must be already allocated by the user, and must be at least *inumParms* size. In what way the motion pointer influences the output? Well, when the motion pointer falls in a determinate cubed zone, delimited, for instance, by 8 vertices

(or pivot points), we assume that each vertex has associated a different snapshot (i.e. a set of *inumParms* values), well, the output will be the weighted average value of the 8 vertices, calculated according on the distance of the motion pointer from each of the 8 vertices. In the case of a default behavior, when the *iConfigTab* argument is not set, the exact output is calculated by using linear interpolation which is applied to each different parameter of the HVS. Anyway, it is possible to influence this behavior by setting the *iConfigTab* argument to a number of a table whose contents can affect one or more HVS parameters. The *iConfigTab* table elements are associated to each HVS parameter and their values affect the HVS output in the following way:

- If *iConfigTab* is equal to -1, corresponding output is skipped, i.e. the element is not calculated, leaving corresponding element value in the *iOutTab* unchanged;
- If *iConfigTab* is equal to zero, then the normal HVS output is calculated (by using weighted average of the nearest vertex of current zone where it falls the motion pointer);
- If *iConfigTab* element is equal to an integer number > zero, then the contents of a table having that number is used as a shape of a table-based interpolation.

## See Also

*hvs1*, *hvs2*, *vphaseseg*

## Credits

Author: Gabriel Maldonado

New in version 5.06

# i

i — Retourne un équivalent de taux-i d'un argument de taux-k.

## Description

Retourne un équivalent de taux-i d'un argument de taux-k.

## Syntaxe

`i(x)` (arguments de taux-k seulement)

Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.



### Note

L'utilisation de `i()` avec un argument expression de taux-k n'est pas recommandée et peut produire des résultats inattendus.

## Voir Aussi

*a, k, abs, exp, frac, int, log, log10, sqrt*

# ibetarand

ibetarand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.

# ibexprnd

ibexprnd — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.



# icauchy

icauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

# ictrl14

ictrl14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl14*.

# ictrl21

ictrl21 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl21*.

# ictrl7

ictrl7 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl7*.

# iexprand

iexprand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.

# if

if — Branchement conditionnel à l'initialisation ou durant l'exécution.

## Description

*if...igoto* -- branchement conditionnel à l'initialisation, dépendant de la valeur de vérité de l'expression logique *ia R ib*. Le branchement n'a lieu que si le résultat est vrai.

*if...kgoto* -- branchement conditionnel durant l'exécution, dépendant de la valeur de vérité de l'expression logique *ka R kb*. Le branchement n'a lieu que si le résultat est vrai.

*if...goto* -- combinaison des deux versions ci-dessus. La condition est testée à chaque passage.

*if...then* -- donne la possibilité de spécifier des blocs conditionnels *if/else/endif*. Tous les blocs *if...then* doivent se terminer par une instruction *endif*. Les instructions *elseif* et *else* sont facultatives. On peut utiliser n'importe quel nombre d'instructions *elseif*. Il ne peut y avoir qu'une seule instruction *else* et elle doit être la dernière instruction conditionnelle avant l'instruction *endif*. Des blocs imbriqués de *if...then* sont permis.



### Note

Notez que si la condition utilise une variable de taux-k (par exemple, « if kval > 0 »), l'instruction *if...goto* ou *if...then* sera ignorée lors de la phase d'initialisation. Cela permet une initialisation de l'opcode même si la variable de taux-k a déjà reçu une valeur appropriée par une instruction init antérieure.

## Syntaxe

```
if ia R ib igoto label
```

```
if ka R kb kgoto label
```

```
if xa R xb goto label
```

```
if xa R xb then
```

où *label* est dans le même bloc d'instrument et n'est pas une expression, et où *R* est un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

Si l'on utilise *goto* ou *then* à la place de *kgoto* ou *igoto*, le comportement est déterminé par le type étant comparé. Si la comparaison utilise des variables de taux-k, *kgoto* est utilisé et vice-versa.



### Note

Les instructions *If/then/goto* ne peuvent pas effectuer de comparaisons de type audio. On ne peut pas mettre de variables de type-a dans les expressions de comparaison pour ces opcodes. La raison en est que les variables audio sont des vecteurs qui ne peuvent pas être comparés de la même façon que des scalaires. Si l'on doit comparer des échantillons audio individuellement il faut utiliser *kr = 1* ou *Compareurs et Accumulateurs*

## Exemples

Voici un exemple de la combinaison if ... igoto. Il utilise le fichier *igoto.csd* [examples/igoto.csd].

### Exemple 247. Exemple de la combinaison if ... igoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o igoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the value of the 4th p-field from the score.
iparam = p4

; If iparam is 1 then play the high note.
; If not then play the low note.
if (iparam == 1) igoto highnote
    igoto lownote

highnote:
    ifreq = 880
    goto playit

lownote:
    ifreq = 440
    goto playit

playit:
; Print the values of iparam and ifreq.
    print iparam
    print ifreq

    a1 oscil 10000, ifreq, 1
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1: iparam = 0.000
instr 1: ifreq = 440.000
instr 1: iparam = 1.000
instr 1: ifreq = 880.000
```

Voici un exemple de la combinaison if ... kgoto. Il utilise le fichier *kgoto.csd* [examples/kgoto.csd].

### Exemple 248. Exemple de la combinaison if ... kgoto.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o kgoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Change kval linearly from 0 to 2 over
; the period set by the third p-field.
kval line 0, p3, 2

; If kval is greater than or equal to 1 then play the high note.
; If not then play the low note.
if (kval >= 1) kgoto highnote
      kgoto lownote

highnote:
      kfreq = 880
      goto playit

lownote:
      kfreq = 440
      goto playit

playit:
; Print the values of kval and kfreq.
printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

      a1 oscil 10000, kfreq, 1
      out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
```



```
kval = 1.999639, kfreq = 880.000000
```

## Exemples

Voici un exemple de la combinaison if ... then. Il utilise le fichier *ifthen.csd* [examples/ifthen.csd].

### Exemple 249. Exemple de la combinaison if ... then.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ifthen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the note value from the fourth p-field.
knote = p4

; Does the user want a low note?
if (knote == 0) then
    kcps = 220
; Does the user want a middle note?
elseif (knote == 1) then
    kcps = 440
; Does the user want a high note?
elseif (knote == 2) then
    kcps = 880
endif

; Create the note.
kamp init 25000
ifn = 1
a1 oscili kamp, kcps, ifn

out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4: 0=low note, 1=middle note, 2=high note.
; Play Instrument #1 for one second, low note.
i 1 0 1 0
; Play Instrument #1 for one second, middle note.
i 1 1 1 1
; Play Instrument #1 for one second, high note.
i 1 2 1 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*elseif, else, endif, goto, igoto, kgoto, tigoto, timeout*

## Crédits

Exemples écrits par Kevin Conder.

Note ajoutée par Jim Aikin.

Février 2004. Note ajoutée par Matt Ingalls.

# igauss

igauss — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.

# igoto

igoto — Transfère le contrôle lors de la phase d'initialisation.

## Description

Transfère le contrôle sans condition vers l'instruction étiquetée par *label*, lors de la phase d'initialisation seulement.

## Syntaxe

```
igoto label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.

## Exemples

Voici un exemple de l'opcode igoto. Il utilise le fichier *igoto.csd* [examples/igoto.csd].

### Exemple 250. Exemple de l'opcode igoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o igoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the value of the 4th p-field from the score.
iparam = p4

; If iparam is 1 then play the high note.
; If not then play the low note.
if (iparam == 1) igoto highnote
    igoto lownote

highnote:
ifreq = 880
goto playit

lownote:
ifreq = 440
goto playit

playit:
; Print the values of iparam and ifreq.
print iparam
```

```

    print ifreq

    a1 oscil 10000, ifreq, 1
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

instr 1:  iparam = 0.000
instr 1:  ifreq = 440.000
instr 1:  iparam = 1.000
instr 1:  ifreq = 880.000

```

## Voir Aussi

*cggoto, cigoto, ckgoto, goto, if, kgoto, rigoto, tigoto, timeout*

## Crédits

Example written by Kevin Conder.

# ihold

ihold — Crée une note tenue.

## Description

Transforme une note de durée finie en une note « tenue ».

## Syntaxe

ihold

## Exécution

*ihold* -- cette instruction de la phase d'initialisation transforme une note de durée finie en une note « tenue ». Elle a ainsi le même effet qu'une valeur de p3 négative (voir l'*instruction i* de la partition), sauf qu'ici p3 reste positif et que l'instrument se redéfinit lui-même pour durer indéfiniment. La note peut être arrêtée explicitement par un *turnoff*, ou son espace peut être utilisé par une autre note ayant le même numéro d'instrument (c-à-d qu'elle est liée à cette note). N'agit que pendant la phase d'initialisation ; inopérant pendant une phase de réinitialisation (*reinit*).

## Exemples

Voici un exemple de l'opcode *ihold*. Il utilise le fichier *ihold.csd* [examples/ihold.csd].

### Exemple 251. Exemple de l'opcode *ihold*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac -iadc -d ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ihold.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; A simple oscillator with its note held indefinitely.
a1 oscil 10000, 440, 1
ihold

; If p4 equals 0, turn the note off.
if (p4 == 0) kgoto offnow
kgoto playit

offnow:
; Turn the note off now.
```

```
turnoff

playit:
  ; Play the note.
  out al
endin

</CsInstruments>
<CsScore>

  ; Table #1: an ordinary sine wave.
  f 1 0 32768 10 1

  ; p4 = turn the note off (if it is equal to 0).
  ; Start playing Instrument #1.
  i 1 0 1 1
  ; Turn Instrument #1 off after 3 seconds.
  i 1 3 1 0
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*i Statement, turnoff*

## Crédits

Exemple écrit par Kevin Conder.

# ilinrand

ilinrand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.



# imagecreate

imagecreate — Create an empty image of a given size.

## Description

Create an empty image of a given size. Individual pixel values can then be set with. *imagegetpixel*.

## Syntax

```
iimagenum imagecreate iwidth, iheight
```

## Initialization

*iimagenum* -- number assigned to the created image.

*iwidth* -- desired image width.

*iheight* -- desired image height.

## Examples

Here is an example of the imagecreate opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 252. Example of the imageload opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac        -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
-o imageopcodes.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin

    instr 2
```

```
imagesave giimage2, "imageout.png"  
endin  
  
instr 3  
imagefree giimage1  
imagefree giimage2  
endin  
  
</CsInstruments>  
<CsScore>  
  
i1 1 1  
i2 2 1  
i3 3 1  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## See Also

*imageload, imagesize, imagesave, imagegetpixel, imagesetpixel, imagefree*

## Credits

Author: Cesare Marilungo

New in version 5.08

# imagefree

imagecreate — Frees memory allocated for a previously loaded or created image.

## Description

Frees memory allocated for a previously loaded or created image.

## Syntax

```
imagefree iimagenum
```

## Initialization

*iimagenum* -- reference of the image to free.

## Examples

Here is an example of the imagefree opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 253. Example of the imagefree opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac        -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
endin

instr 2

imagesave giimage2, "imageout.png"
endin

instr 3
imagefree giimage1
```

```
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*imageload, imagecreate, imagesize, imagesave, imagegetpixel, imagesetpixel*

## Credits

Author: Cesare Marilungo

New in version 5.08

# imagegetpixel

imagegetpixel — Return the RGB pixel values of a previously opened or created image.

## Description

Return the RGB pixel values of a previously opened or created image. An image can be loaded with *imageload*. An empty image can be created with *imagecreate*.

## Syntax

```
ared agreen ablue imagegetpixel iimagenum, ax, ay
```

```
kred kgreen kblue imagegetpixel iimagenum, kx, ky
```

## Initialization

*iimagenum* -- the reference of the image.. It should be a value returned by *imageload* or *imagecreate*.

## Performance

*ax* (*kx*) -- horizontal pixel position (must be a float from 0 to 1).

*ay* (*ky*) -- vertical pixel position (must be a float from 0 to 1).

*ared* (*kred*) -- red value of the pixel (mapped to a float from 0 to 1).

*agreen* (*kgreen*) -- green value of the pixel (mapped to a float from 0 to 1).

*ablue* (*kblue*) -- blue value of the pixel (mapped to a float from 0 to 1).

## Examples

Here is an example of the imagegetpixel opcode. It uses the files *imageopcodesdemo2.csd* [examples/imageopcodesdemo2.csd] *test1.png* [examples/test1.png] and *test2.png* [examples/test2.png].

### Exemple 254. Example of the imagegetpixel opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o imageopcodesdemo2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr          =          48000
ksmps       =          100
nchnls      = 2

;By Cesare Marilungo 2008
zakinit 10,1
```

```
;Load the image - should be 512x512 pixels
giimage imageload "test1.png"
;giimage imageload "test2.png" ;--try this too
giimagew, giimageh imagesize giimage

giwave ftgen 1, 0, 1024, 10, 1
gifrqs ftgen 2,0,512,-5, 1,512,10
giamps ftgen 3, 0, 512, 10, 1

    instr 100

kindex = 0
icnt = giimageh
kx_ linseg 0, p3, 1
kenv linseg 0, .2, 500, p3 - .4, 500, .2, 0

; Read a column of pixels and store the red values
; inside the table 'giamps'
loop:
    ky_ = kindex/giimageh

    ;Get the pixel color values at kx_, ky_
    kred, kgreen, kblue imagegetpixel giimage, kx_, ky_

    ;Write the red values inside the table 'giamps'
    tablew kred, kindex, giamps
    kindex = kindex+1

if (kindex < icnt) kgoto loop

; Use an oscillator bank (additive synthesis) to generate sound
; setting amplitudes for each partial according to the image
asig adsynt kenv, 220, giwave, gifrqs, giamps, icnt, 2
outs asig, asig

    endin

    instr 101
    ; Free memory used by the image
imagefree giimage
    endin

</CsInstruments>
<CsScore>

t 0 60

i100 1 20
i101 21 1

e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*imageload, imagecreate, imagesize, imagesave, imagesetpixel, imagefree*

## Credits

Author: Cesare Marilungo

New in version 5.08

# imageload

imageload — Load an image.

## Description

Load an image and return a reference to it. Individual pixel values can then be accessed with *imagegetpixel*.



### Note

The image processing opcodes can only load png images

## Syntax

```
iimagenum imageload filename
```

## Initialization

*iimagenum* -- number assigned to the loaded image.

*filename* -- The filename of the image to load (should be a valid PNG image file).

## Examples

Here is an example of the imageload opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 255. Example of the imageload opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
;-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
```

```
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_  
loop_lt kndx, 0.5, (giimageh), myloop  
    endin  
  
    instr 2  
  
imagesave giimage2, "imageout.png"  
    endin  
  
    instr 3  
imagefree giimage1  
imagefree giimage2  
    endin  
  
</CsInstruments>  
<CsScore>  
  
i1 1 1  
i2 2 1  
i3 3 1  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## See Also

*imagecreate, imagesize, imagesave, imagegetpixel, imagesetpixel, imagefree*

## Credits

Author: Cesare Marilungo

New in version 5.08



# imagesave

imagesave — Save a previously created image.

## Description

Save a previously created image. An empty image can be created with *imagecreate* and its pixel RGB values can be set with *imagesetpixel*. The image will be saved in PNG format.

## Syntax

```
imagesave iimagenum, filename
```

## Initialization

*iimagenum* -- the reference of the image to be save. It should be a value returned by *imagecreate*.

*filename* -- The filename to use to save the image.

## Examples

Here is an example of the imagesave opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 256. Example of the imagesave opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
;-odac       -iadc    ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
```

```
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*imageload, imagecreate, imagesize, imagegetpixel, imagesetpixel, imagefree*

## Credits

Author: Cesare Marilungo

New in version 5.08

# imagesetpixel

imagesetpixel — Set the RGB value of a pixel inside a previously opened or created image.

## Description

Set the RGB value of a pixel inside a previously opened or created image. An image can be loaded with *imageload*. An empty image can be created with *imagecreate* and saved with *imagesave*.

## Syntax

```
imagegetpixel iimagenum, ax, ay, ared, agreen, ablue
```

```
imagegetpixel iimagenum, kx, ky, kred, kgreen, kblue
```

## Initialization

*iimagenum* -- the reference of the image.. It should be a value returned by *imageload* or *imagecreate*.

## Performance

*ax (kx)* -- horizontal pixel position (must be a float from 0 to 1).

*ay (ky)* -- vertical pixel position (must be a float from 0 to 1).

*ared (kred)* -- red value of the pixel (mapped to a float from 0 to 1).

*agreen (kgreen)* -- green value of the pixel (mapped to a float from 0 to 1).

*ablue (kblue)* -- blue value of the pixel (mapped to a float from 0 to 1).

## Examples

Here is an example of the imagesetpixel opcode. It uses the file *imageopcodes.csd* [examples/ima-geopcodes.csd].

### Exemple 257. Example of the imagesetpixel opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
;-odac       -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
```

```

giimagew, giimageh imagesize giimagel
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimagel, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimagel
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>

```

## See Also

*imageload, imagecreate, imagesize, imagesave, imagegetpixel, imagefree*

## Credits

Author: Cesare Marilungo

New in version 5.08

# imagesize

imagesize — Return the width and height of a previously opened or created image.

## Description

Return the width and height of a previously opened or created image. An image can be loaded with *imageload*. An empty image can be created with *imagecreate*.

## Syntax

```
iwidth iheight imagesize iimagenum
```

## Initialization

*iimagenum* -- the reference of the image.. It should be a value returned by *imageload* or *imagecreate*.

*iwidth* -- image width.

*iheight* -- image height.

## Examples

Here is an example of the imagesize opcode. It uses the file *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 258. Example of the imagesize opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
;-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o imageopcodes.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1

myloop:
ky_ = kndx/(giimageh)
kr_ kg_ kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, (giimageh), myloop
endin
```

```
instr 2
imagesave giimage2, "imageout.png"
endin

instr 3
imagefree giimage1
imagefree giimage2
endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*imageload, imagecreate, imagesave, imagegetpixel, imagesetpixel, imagefree*

## Credits

Author: Cesare Marilungo

New in version 5.08

# imidic14

imidic14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic14*.

# imidic21

imidic21 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic21*.



# imidic7

imidic7 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic7*.

# in

in — Lit des données audio mono depuis un périphérique externe ou un flot.

## Description

Lit des données audio mono depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls*=1. Avec des orchestres dont *nchnls* > 1, la sortie audio ne sera pas correcte.

## Syntaxe

`arl in`

## Exécution

Lit des données audio mono depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Voir Aussi

*diskin, inh, inh, ino, inq, ins, soundin*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

Déjà dans la version 3.30

# in32

in32 — Lit un signal audio sur 32 canaux depuis un périphérique externe ou un flot.

## Description

Lit un signal audio sur 32 canaux depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls*=32. Avec des orchestres dont *nchnls* > 32, la sortie audio ne sera pas correcte.

## Syntaxe

```
ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \  
    ar15, ar16, ar17, ar18, ar19, ar20, ar21, ar22, ar23, ar24, ar25, ar26, \  
    ar27, ar28, ar29, ar30, ar31, ar32 in32
```

## Exécution

*in32* lit un signal audio sur 32 canaux depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne.

## Voir Aussi

*inch*, *inx*, *inz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# inch

inch — Lit depuis un canal numéroté d'un signal audio externe ou d'un flot.

## Description

Lit depuis un canal numéroté d'un signal audio externe ou d'un flot.

## Syntaxe

```
ain inch kchan
```

## Exécution

*ain* - signal audio en entrée

*kchan* - numéro du canal

*inch* lit depuis un canal numéroté déterminé par *kchan* vers *ain*. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son). On peut aussi utiliser *inch* pour recevoir des données audio en temps réel depuis l'interface audio au moyen de *-iadc*.



### Note

La plus grande valeur du paramètre *kchan* de *inch* dépend de *nchnls*. Si *kchan* est supérieur à *nchnls*, *ain* restera silencieux. Noter que dans ce cas *inch* donnera un avertissement et pas une erreur.

## Voir Aussi

*in32*, *inx*, *inz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# inh

inh — Lit des données audio sur six canaux depuis un périphérique externe ou un flot.

## Description

Lit des données audio sur six canaux depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls*=6. Avec des orchestres dont *nchnls* > 6, la sortie audio ne sera pas correcte.

## Syntaxe

*ar1*, *ar2*, *ar3*, *ar4*, *ar5*, *ar6* **inh**

## Exécution

Lit des données audio sur six canaux depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Voir Aussi

*diskin*, *in*, *ino*, *inq*, *ins*, *soundin*

## Crédits

Auteur : John ffitch

# init

init — Met la valeur de l'expression de taux-i dans une variable de taux-k ou de taux-a.

## Syntaxe

```
ares init iarg  
  
ires init iarg  
  
kres init iarg  
  
ares, ... init iarg, ...  
  
ires, ... init iarg, ...  
  
kres, ...init iarg, ...
```

## Description

Met la valeur de l'expression de taux-i dans une variable de taux-k ou de taux-a.

## Initialisation

Met la valeur de l'expression de taux-i *iarg* dans une variable de taux-k ou de taux-a, c-à-d., initialise le résultat. Noter que **init** présente le seul cas d'une instruction de la période d'initialisation autorisée à écrire dans un résultat de la période d'exécution (taux-k ou -a) ; cette instruction n'a aucun effet pendant l'exécution.

Depuis la version 5.13 il est possible d'initialiser jusqu'à 24 variables de la même classe dans une instruction. S'il y a plus de variables en sortie que d'expressions en entrée, la dernière expression est répétée. C'est une erreur d'avoir plus d'entrées que de sorties.

## Voir Aussi

*=, divz, tival*

## Crédits

*init* était présent dans le Csound original, mais l'extension aux valeurs multiples a été ajoutée par

Auteur : John ffitch  
Université de Bath, and Codemist Ltd.  
Bath, UK  
February 2010

Nouveau dans la version 5.13

# initc14

initc14 — Initialise les contrôleurs pour créer une valeur MIDI sur 14 bit.

## Description

Initialise les contrôleurs pour créer une valeur MIDI sur 14 bit.

## Syntaxe

```
initc14 ichan, ictlno1, ictlno2, ivalue
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*ivalue* -- valeur décimale (doit être entre 0 et 1)

## Exécution

*initc14* peut être utilisé conjointement avec les opcodes *midic14* et *ctrl14* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic14* et de *ctrl14*:

$$ivalue = (valeur\_initiale - min) / (max - min)$$

## Voir Aussi

*ctrl7*, *ctrl14*, *ctrl21*, *ctrlinit*, *initc7*, *initc21*, *midic7*, *midic14*, *midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# initc21

initc21 — Initialise les contrôleurs pour créer une valeur MIDI sur 21 bit.

## Description

Initialise les contrôleurs pour créer une valeur MIDI sur 21 bit.

## Syntaxe

```
initc21 ichan, ictlno1, ictlno2, ictlno3, ivalue
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids moyen (0-127)

*ictlno3* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*ivalue* -- valeur décimale (doit être entre 0 et 1)

## Exécution

*initc21* peut être utilisé conjointement avec les deux opcodes *midic21* et *ctrl21* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic21* et de *ctrl21*:

$$ivalue = (valeur\_initiale - min) / (max - min)$$

## Voir aussi

*ctrl7*, *ctrl14*, *ctrl21*, *ctrlinit*, *initc7*, *initc14*, *midic7*, *midic14*, *midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.



# initc7

*initc7* — Initialise le contrôleur utilisé pour créer une valeur MIDI sur 7 bit.

## Description

Initialise le contrôleur MIDI *ictlno* avec *ivalue*

## Syntaxe

```
initc7 ichan, ictlno, ivalue
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlno* -- numéro du contrôleur (0-127)

*ivalue* -- valeur décimale (doit être entre 0 et 1)

## Exécution

*initc7* peut être utilisé conjointement avec les opcodes *midic7* et *ctrl7* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic7* et de *ctrl7*:

$$ivalue = (valeur\_initiale - min) / (max - min)$$

## Voir aussi

*ctrl7*, *ctrl14*, *ctrl21*, *ctrlinit*, *initc14*, *initc21*, *midic7*, *midic14*, *midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# inleta

inleta — Receives an arate signal into an instrument through a named port.

## Description

Receives an arate signal into an instrument through a named port.

## Syntax

`asignal inleta Sname`

## Initialization

*Sname* -- String name of the inlet port. The name of the inlet is implicitly qualified by the instrument name or number, so it is valid to use the same inlet name in more than one instrument (but not to use the same inlet name twice in one instrument).

## Performance

*asignal* -- audio input signal

During performance, the arate inlet signal is received from each instance of an instrument containing an outlet port to which this inlet has been connected using the connect opcode. The signals of all the outlets connected to an inlet are summed in the inlet.

## See Also

*outleta outletk outletf inletk inletf connect alwayson ftgenonce*

## Credits

By: Michael Gogins 2009

# inletk

inletk — Receives a krate signal into an instrument from a named port.

## Description

Receives a krate signal into an instrument from a named port.

## Syntax

`ksignal inletk Sname`

## Initialization

*Sname* -- String name of the inlet port. The name of the inlet is implicitly qualified by the instrument name or number, so it is valid to use the same inlet name in more than one instrument (but not to use the same inlet name twice in one instrument).

## Performance

*ksignal* -- krate input signal

During performance, the krate inlet signal is received from each instance of an instrument containing an outlet port to which this inlet has been connected using the connect opcode. The signals of all the outlets connected to an inlet are summed in the inlet.

## See Also

*outleta outletk outletf inleta inletf connect alwayson ftgenonce*

## Credits

By: Michael Gogins 2009

# inletf

inletf — Receives an frate signal (fsig) into an instrument from a named port.

## Description

Receives an frate signal (fsig) into an instrument from a named port.

## Syntax

`fsignal inletf Sname`

## Initialization

*Sname* -- String name of the inlet port. The name of the inlet is implicitly qualified by the instrument name or number, so it is valid to use the same inlet name in more than one instrument (but not to use the same inlet name twice in one instrument).

## Performance

*ksignal* -- krate input signal

During performance, the frate inlet signal is received from each instance of an instrument containing an outlet port to which this inlet has been connected using the connect opcode. The signals of all the outlets connected to an inlet are combined in the inlet.

## See Also

*outleta outletk outletf inleta inletk connect alwayson ftgenonce*

## Credits

By: Michael Gogins 2009

# ino

ino — Lit des données audio sur huit canaux depuis un périphérique externe ou un flot.

## Description

Lit des données audio sur huit canaux depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls*=8. Avec des orchestres dont *nchnls* > 8, la sortie audio ne sera pas correcte.

## Syntaxe

*ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8 ino*

## Exécution

Lit des données audio sur huit canaux depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Voir Aussi

*diskin, in, inh, inq, ins, soundin*

## Crédits

Auteur : John ffitch

# inq

inq — Lit des données audio quadro depuis un périphérique externe ou un flot.

## Description

Lit des données audio quadro depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls*=4. Avec des orchestres dont *nchnls* > 4, la sortie audio ne sera pas correcte.

## Syntaxe

```
ar1, ar2, ar3, a4 inq
```

## Exécution

Lit des données audio quadro depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Voir Aussi

*diskin, in, inh, inh, ino, ins, soundin*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# inrg

**inrg** — Permet une entrée depuis un ensemble de canaux contigus du périphérique d'entrée audio.

## Description

*inrg* lit des données audio depuis un ensemble de canaux contigus du périphérique d'entrée audio.

## Syntaxe

```
inrg kstart, ain1 [,ain2, ain3, ..., ainN]
```

## Exécution

*kstart* - le numéro du premier canal du périphérique d'entrée à lire (les numéros des canaux commencent à 1, qui est le premier canal).

*ain1*, *ain2*, ... *ainN* - les arguments de sortie remplis avec les données audio lues depuis les canaux correspondants.

*inrg* permet une entrée depuis un ensemble de canaux contigus du périphérique d'entrée audio. *kstart* indique le premier canal à lire (le canal 1 étant le premier canal). Il faut s'assurer que le nombre obtenu en ajoutant à *kstart* le nombre de canaux à lire - 1 est  $\leq nchnls$ .



### Note

Noter que cet opcode est exceptionnel en ce sens qu'il produit sa « sortie » dans les paramètres situés à sa droite.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# ins

ins — Lit des données audio stéréo depuis un périphérique externe ou un flot.

## Description

Lit des données audio stéréo depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls*=2. Avec des orchestres dont *nchnls* > 2, la sortie audio ne sera pas correcte.

## Syntaxe

`ar1, ar2 ins`

## Exécution

Lit des données audio stéréo depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Voir Aussi

*diskin, in, inh, inh, ino, inq, soundin mp3in,*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997



# insremot

**insremot** — An opcode which can be used to implement a remote orchestra. This opcode will send note events from a source machine to one destination.

## Description

With the `insremot` and `insglobal` opcodes you are able to perform instruments on remote machines and control them from a master machine. The remote opcodes are implemented using the master/client model. All the machines involved contain the same orchestra but only the master machine contains the information of the score. During the performance the master machine sends the note events to the clients. The `insremot` opcode will send events from a source machine to one destination if you want to send events to many destinations (broadcast) use the `insglobal` opcode instead. These two opcodes can be used in combination.

## Syntax

```
insremotdestination, isource, instrnum [,instrnum...]
```

## Initialization

*destination* -- a string that is the intended host computer (e.g. 192.168.0.100). This is the destination host which receives the events from the given instrument.

*isource* -- a string that is the intended host computer (e.g. 192.168.0.100). This is the source host which generates the events of the given instrument and sends it to the address given by *destination*.

*instrnum* -- a list of instrument numbers which will be played on the destination machine

## Examples

Here is an example of the `insremot` opcode. It uses the files `insremot.csd` [examples/insremot.csd] and `insremotM.csd` [examples/insremotM.csd].

### Exemple 259. Example of the insremot opcode.

The simple example below shows the `bilbar` example played on a remote machine. The master machine is named "192.168.1.100" and the client "192.168.1.101". Start the client on the machine (it will wait to receive the events from the master machine) and then start the master. Here is the command on linux to start a client (`csound -+rtaudio=alsa -odac -dm0 insremot.csd`), and the command on the master machine will look like this (`csound -+rtaudio=alsa -odac -dm0 insremotM.csd`).

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o insremot.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>
nchnls = 1

insremot "192.168.1.100", "192.168.1.101", 1

instr 1
  aq barmodel 1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
  out aq
endin

</CsInstruments>
<CsScore>
f0 360

e
</CsScore>
</CsoundSynthesizer>

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac -iadc ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o insremotM.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
nchnls = 1

insremot "192.168.1.100", "192.168.1.101", 1

instr 1
  aq barmodel 1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
  out aq
endin

</CsInstruments>
<CsScore>
i1 0 0.5 3 0.2 500 0.05
i1 0.5 0.5 -3 0.3 1000 0.05
i1 1.0 0.5 -3 0.4 1000 0.1
i1 1.5 4.0 -3 0.5 800 0.05
e
</CsScore>
</CsoundSynthesizer>

```

## See also

*insglobal, midglobal, midremot, remoteport*

## Credits

Author: Simon Schampijer  
2006

New in version 5.03

# insglobal

*insglobal* — An opcode which can be used to implement a remote orchestra. This opcode will send note events from a source machine to many destinations.

## Description

With the *insremot* and *insglobal* opcodes you are able to perform instruments on remote machines and control them from a master machine. The remote opcodes are implemented using the master/client model. All the machines involved contain the same orchestra but only the master machine contains the information of the score. During the performance the master machine sends the note events to the clients. The *insglobal* opcode sends the events to all the machines involved in the remote concert. These machines are determined by the *insremot* definitions made above the *insglobal* command. To send events to only one machine use *insremot*.

## Syntax

```
insglobalsource, instrnum [,instrnum...]
```

## Initialization

*isource* -- a string that is the intended host computer (e.g. 192.168.0.100). This is the source host which generates the events of the given instrument(s) and sends it to all the machines involved in the remote concert.

*instrnum* -- a list of instrument numbers which will be played on the destination machines

## Examples

See the entry for *insremot* for an example of usage.

## See also

*insremot*, *midglobal*, *midremot*, *remoteport*

## Credits

Author: Simon Schampijer  
2006

New in version 5.03

# instimek

instimek — Obsolète.

## Description

Obsolète depuis la version 3.62. Utiliser plutôt l'opcode *timeinstk*.

## Crédits

David M. Boothe est à l'origine du signalement de ce nom obsolète.

# instimes

instimes — Obsolète.

## Description

Obsolète depuis la version 3.62. Utiliser plutôt l'opcode *timeinsts*.

## Crédits

David M. Boothe est à l'origine du signalement de ce nom obsolète.

# instr

instr — Commence un bloc d'instrument.

## Description

Commence un bloc d'instrument.

## Syntaxe

```
instr i, j, ...
```

## Initialisation

Commence un bloc d'instrument, définissant les instruments *i, j, ...*

*i, j, ...* doivent être des nombres, pas des expressions. Tout entier positif convient, dans n'importe quel ordre, mais on préfère éviter les nombres excessivement grands.



### Note

Il peut y avoir n'importe quel nombre de blocs d'instrument dans un orchestre.

On peut définir les instruments dans n'importe quel ordre (mais ils seront toujours initialisés et exécutés par ordre de numéro d'instrument ascendant, à l'exception des notes provoquées par des événements en temps réel, qui sont initialisées dans l'ordre où elles sont reçues mais toujours exécutées par ordre de numéro d'instrument ascendant). Les blocs d'instruments ne peuvent pas être imbriqués (un bloc ne peut pas en contenir un autre).

## Exécution

### Appeler un Instrument depuis un Instrument



### Avertissement

Ce comportement n'est pas complètement disponible dans Csound 5. Dans Csound 5, il faut utiliser l'opcode *subinstr*.

On peut appeler un instrument depuis un instrument comme si c'était un opcode soit au moyen de l'opcode *subinstr* soit en spécifiant un instrument avec un nom textuel :

```
instr MonOscil  
...  
endin
```

Par défaut, tous les paramètres de sortie correspondent aux sorties de l'instrument exprimées par des opcodes de *sortie de signal*. Tous les paramètres d'entrée sont affectés aux p-champs de l'instrument appelé en commençant par le quatrième, p4. Les valeurs des deuxième et troisième p-champs de l'instrument

appelé, p2 et p3, sont automatiquement fixés à la même valeur que ceux de l'instrument appelant.

Un instrument nommé doit être défini avant les instrument qui l'appellent.



## Conseils

Si vous utiliser l'opcode *outc*, vous pouvez créer un instrument qui pourra être compilé et fonctionner dans des orchestres avec n'importe quel nombre de canaux plus grand au égal ou nombre de canaux de sortie de cet instrument.

Il est intéressant d'utiliser la fonctionnalité *#include* avec les instruments nommés. Vous pouvez définir vos instruments nommés dans des fichiers séparés, et utiliser *#include* lorsque vous en avez besoin.

## Exemples

Voici un exemple de l'opcode *instr*. Il utilise le fichier *instr.csd* [examples/instr.csd].

### Exemple 260. Exemple de l'opcode *instr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o instr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  a1 oscils iamp, icps, iphs
  out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*endin, in, out, opcode, endop, setksmps, xin, xout, subinstr, subinstrinit*

## Crédits

Exemple écrit par Kevin Conder.



# int

int — Extrait la partie entière d'un nombre décimal.

## Description

Retourne la partie entière de  $x$ .

## Syntaxe

```
int(x) (taux-i ou taux-k ; fonctionne aussi au taux-a dans Csound5)
```

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode int. Il utilise le fichier *int.csd* [examples/int.csd].

### Exemple 261. Exemple de l'opcode int.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o int.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
i1 = 16 / 5
i2 = int(i1)

print i2
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i2 = 3.000
```

## Voir Aussi

*abs, exp, frac, log, log10, i, sqrt*

## Crédits

Exemple écrit par Kevin Conder.

# integ

integ — Modify a signal by integration.

## Description

Modify a signal by integration.

## Syntax

```
ares integ asig [, iskip]
```

```
kres integ ksig [, iskip]
```

## Initialization

*iskip* (optional) -- initial disposition of internal save space (see *reson*). The default value is 0.

## Performance

*integ* and *diff* perform integration and differentiation on an input control signal or audio signal. Each is the converse of the other, and applying both will reconstruct the original signal. Since these units are special cases of low-pass and high-pass filters, they produce a scaled (and phase shifted) output that is frequency-dependent. Thus *diff* of a sine produces a cosine, with amplitude  $2 * \sin(\pi * Hz / sr)$  that of the original (for each component partial); *integ* will inversely affect the magnitudes of its component inputs. With this understanding, these units can provide useful signal modification.

## Examples

Here is an example of the integ opcode. It uses the file *integ.csd* [examples/integ.csd].

### Exemple 262. Example of the integ opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o integ.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 -- a differentiated signal.
instr 1
; Generate a band-limited pulse train.
```

```

asrc buzz 20000, 440, 20, 1

; Differentiate the signal.
adiff diff asrc

out adiff
endin

; Instrument #2 -- a re-integrated signal.
instr 2
; Generate a band-limited pulse train.
asrc buzz 20000, 440, 20, 1

; Differentiate the signal.
adiff diff asrc

; Re-integrate the previously differentiated signal.
al integ adiff

out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>

```

## See Also

*diff, downsamp, interp, samphold, upsamp*

## Credits

Example written by Kevin Conder.

# interp

interp — Converts a control signal to an audio signal using linear interpolation.

## Description

Converts a control signal to an audio signal using linear interpolation.

## Syntax

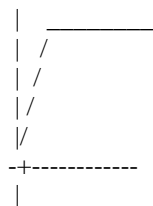
```
ares interp ksig [, iskip] [, imode]
```

## Initialization

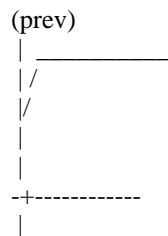
*iskip* (optional, default=0) -- initial disposition of internal save space (see *reson*). The default value is 0.

*imode* (optional, default=0) -- sets the initial output value to the first k-rate input instead of zero. The following graphs show the output of interp with a constant input value, in the original, when skipping init, and in the new mode:

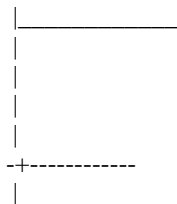
### Exemple 263. iskip=0, imode=0



### Exemple 264. iskip=1, imode=0



### Exemple 265. iskip=0, imode=1



## Performance

*ksig* -- input k-rate signal.

*interp* converts a control signal to an audio signal. It uses linear interpolation between successive kvals.

## Examples

Here is an example of the *interp* opcode. It uses the file *interp.csd* [examples/interp.csd].

### Exemple 266. Example of the *interp* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o interp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 8000
kr = 8
ksmps = 1000
nchnls = 1

; Instrument #1 - a simple instrument.
instr 1
; Create an amplitude envelope.
kamp linseg 0, p3/2, 20000, p3/2, 0

; The amplitude envelope will sound rough because it
; jumps every ksmps period, 1000.
a1 oscil kamp, 440, 1
out a1
endin

; Instrument #2 - a smoother sounding instrument.
instr 2
; Create an amplitude envelope.
kamp linseg 0, p3/2, 25000, p3/2, 0
aamp interp kamp
```

```
    ; The amplitude envelope will sound smoother due to
    ; linear interpolation at the higher a-rate, 8000.
    al oscil aamp, 440, 1
    out al
endin

</CsInstruments>
<CsScore>

    ; Table #1, a sine wave.
    f 1 0 256 10 1

    ; Play Instrument #1 for two seconds.
    i 1 0 2
    ; Play Instrument #2 for two seconds.
    i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*diff, downsamp, integ, samphold, upsamp*

## Credits

Example written by Kevin Conder.

Updated November 2002, thanks to a note from both Rasmus Ekman and Istvan Varga.

# invalue

invalue — Lit un signal de taux-k depuis un canal défini par l'utilisateur.

## Description

Lit un signal de taux-k ou une chaîne de caractères depuis un canal défini par l'utilisateur.

## Syntaxe

```
kvalue invalue "channel name"
```

```
Sname invalue "channel name"
```

## Exécution

*kvalue* -- La valeur de taux-k lue depuis le canal.

*Sname* -- La variable chaîne de caractères lue depuis le canal.

*"channel name"* -- Un entier, une chaîne de caractères (entre guillemets), ou une variable chaîne de caractères identifiant le canal.

## Voir Aussi

*outvalue*

## Crédits

Auteur : Matt Ingalls



# inx

inx — Lit des données audio sur 16 canaux depuis un périphérique externe ou un flot.

## Description

Lit des données audio sur 16 canaux depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls*=16. Avec des orchestres dont *nchnls* > 16, la sortie audio ne sera pas correcte.

## Syntaxe

```
ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, \  
ar13, ar14, ar15, ar16 inx
```

## Exécution

*inx* lit un signal audio sur 16 canaux depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne.

## Voir Aussi

*in32*, *inch*, *inz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# inz

*inz* — Lit des échantillons audio multi-canaux depuis un périphérique externe ou un flot vers un tableau ZAK.

## Description

Lit des échantillons audio multi-canaux depuis un périphérique externe ou un flot vers un tableau ZAK.

## Syntaxe

```
inz ksigl
```

## Exécution

*inz* lit des échantillons audio sur *nchnls* vers un tableau ZAK commençant à *ksigl*. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne.

## Voir Aussi

*in32*, *inch*, *inx*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# ioff

ioff — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteoff*.

# ion

ion — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteon*.

# iondur

iondur — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteondur*.

# iondur2

iondur2 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteondur2*.

# ioutat

ioutat — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outiat*.

# ioutc

ioutc — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outic*.



# ioutc14

ioutc14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outic14*.

# ioutpat

ioutpat — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipat*.

# ioutpb

ioutpb — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipb*.

# ioutpc

ioutpc — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipc*.

# ipcauchy

ipcauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

# ipoisson

ipoisson — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

# ipow

ipow — Obsolète.

## Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

# is16b14

is16b14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *s16b14*.



# is32b14

is32b14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *s32b14*.

# islider16

islider16 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider16*.

# islider32

islider32 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider32*.

# islider64

islider64 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider64*.

# islider8

islider8 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider8*.

# itablecopy

itablecopy — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tablecopy*.

# itablegpw

itablegpw — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableigpw*.

# itablemix

itablemix — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableimix*.



# itablew

itablew — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableiw*.

# itrirand

itrirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.

# iunirand

iunirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.

# iweibull

iweibull — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

# jacktransport

jacktransport — Start/stop jack\_transport and can optionally relocate the playback head.

## Description

Start/stop jack\_transport and can optionally relocate the playback head.

## Syntax

```
jacktransport icommand [, ilocation]
```

## Initialization

*icommand* -- 1 to start playing, 0 to stop.

*ilocation* -- optional location in seconds to specify where the playback head should be moved. If omitted, the transport is started from current location.



### Note

Since *jacktransport* depends on jack audio connection kit, it will work only on Linux or Mac OS X systems which have the jack server running.

## Examples

Here is a simple example of the jacktransport opcode. It uses the file *jacktransport.csd* [examples/jacktransport.csd].

### Exemple 267. Simple example of the jacktransport opcode.

```
<CsoundSynthesizer>
<CsOptions>

+rtaudio=JACK -b 64 --sched -o dac:system:playback_
</CsOptions>
<CsInstruments>

sr          =          44100
ksmps       =          16
nchnls      = 2

    instr 1
jacktransport p4, p5
    endin

    instr 2
jacktransport p4
    endin

</CsInstruments>
<CsScore>

i2 0 5 1; play
i2 5 1 0; stop
```

```
i1 6 5 1 2 ; move at 2 seconds and start playing back  
i1 11 1 0 0 ; stop and rewind  
  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## Credits

Author: Cesare Marilungo

New in version 5.08

# jitter

jitter — Génère aléatoirement une suite de segments de droite.

## Description

Génère aléatoirement une suite de segments de droite.

## Syntaxe

kout **jitter** kamp, kcpsMin, kcpsMax

## Exécution

*kamp* -- Amplitude de la déviation de jitter

*kcpsMin* -- Vitesse minimale des variations aléatoires de fréquence (exprimée en cps)

*kcpsMax* -- Vitesse maximale des variations aléatoires de fréquence (exprimée en cps)

*jitter* génère aléatoirement une suite de segments de droite entre *-kamp* et *+kamp*. La durée de chaque segment est une valeur aléatoire générée en fonction des valeurs *kcpsmin* et *kcpsmax*.

On peut utiliser *jitter* pour donner plus de naturel et une « touche analogique » à des sons statiques et monotones. Pour de meilleurs résultats il est conseillé de garder une amplitude modérée.

## Exemples

Voici un exemple de l'opcode jitter. Il utilise le fichier *jitter.csd* [examples/jitter.csd].

### Exemple 268. Exemple de l'opcode jitter.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac       -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o jitter.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1 -- plain instrument.
instr 1
  aplain vco 20000, 220, 2, 0.83

  outs aplain, aplain
endin
```

```
; Instrument #2 -- instrument with jitter.
instr 2
; Create a signal modulated the jitter opcode.
kamp init 2
kcpsmin init 4
kcpsmax init 6
kj jitter kamp, kcpsmin, kcpsmax

aplain vco 20000, 220, 2, 0.83
ajitter vco 20000, 220+kj, 2, 0.83

outs aplain, ajitter
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 3 seconds.
i 1 0 3
; Play Instrument #2 for 3 seconds.
i 2 3 3
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*jitter2, vibr, vibrato*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15



# jitter2

jitter2 — Génère aléatoirement une suite de segments de droite contrôlables par l'utilisateur.

## Description

Génère aléatoirement une suite de segments de droite contrôlables par l'utilisateur.

## Syntaxe

```
kout jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, kamp3, kcps3
```

## Exécution

*ktotamp* -- Amplitude résultante de jitter2

*kamp1* -- Amplitude du premier composant de jitter

*kcps1* -- Vitesse de la variation aléatoire du premier composant de jitter (exprimée en cps)

*kamp2* -- Amplitude du second composant de jitter

*kcps2* -- Vitesse de la variation aléatoire du second composant de jitter (exprimée en cps)

*kamp3* -- Amplitude du troisième composant de jitter

*kcps3* -- Vitesse de la variation aléatoire du troisième composant de jitter (exprimée en cps)

*jitter2* génère une ligne segmentée comme *jitter*, mais ici le résultat est semblable à la somme de trois opcodes *randi*, chacun avec ses propres valeurs d'amplitude et de fréquence (voir *randi* pour plus de détails), qui sont modifiables au taux-k. On peut obtenir différents effets en variant les arguments en entrée.

On peut utiliser *jitter2* pour donner plus de naturel et une « touche analogique » à des sons statiques et monotones. Pour de meilleurs résultats il est conseillé de garder une amplitude modérée.

## Exemples

Voici un exemple de l'opcode jitter2. Il utilise le fichier *jitter2.csd* [examples/jitter2.csd].

### Exemple 269. Exemple de l'opcode jitter2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac       -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o jitter2.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1 -- plain instrument.
instr 1
  aplain vco 20000, 220, 2, 0.83

  outs aplain, aplain
endin

; Instrument #2 -- instrument with jitter.
instr 2
  ; Create a signal modulated with the jitter2 opcode.
  ktotamp init 2
  kamp1 init 0.66
  kcps1 init 3
  kamp2 init 0.66
  kcps2 init 3
  kamp3 init 0.66
  kcps3 init 3
  kj jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, \
    kamp3, kcps3

  aplain vco 20000, 220, 2, 0.83
  ajitter vco 20000, 220+kj, 2, 0.83

  outs aplain, ajitter
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 3 seconds.
i 1 0 3
; Play Instrument #2 for 3 seconds.
i 2 3 3
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*jitter, vibr, vibrato*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15

# jspline

jspline — Un générateur de spline avec gigue.

## Description

Un générateur de spline avec gigue.

## Syntaxe

```
ares jspline xamp, kcpsMin, kcpsMax
```

```
kres jspline kamp, kcpsMin, kcpsMax
```

## Exécution

*kres, ares* -- Signal de sortie.

*xamp* -- Facteur d'amplitude.

*kcpsMin, kcpsMax* -- Intervalle de définition du taux de génération des points. Les limites minimale et maximale sont exprimées en Hz.

*jspline* (générateur de spline avec gigue) génère une courbe lisse basée sur des points aléatoires engendrés au taux [*cpsMin, cpsMax*]. Cet opcode est semblable à *randomi* ou à *randi* ou à *jitter*, toutefois les segments ne sont pas des lignes droites, mais des courbes splines cubiques. Les valeurs de sortie sont approximativement comprises entre *-xamp* et *xamp*. Dans la réalité, l'intervalle peut être un peu plus grand, à cause des courbes d'interpolation entre chaque paire de points aléatoires.

Actuellement les courbes générées sont assez lisses quand *cspMin* n'est pas trop différent de *cpsMax*. Quand l'intervalle *cpsMin-cpsMax* est grand, quelques petites discontinuités peuvent se produire, mais, dans la plupart des cas, cela ne devrait pas poser de problème. L'algorithme sera peut-être amélioré dans les prochaines versions.

Ces opcodes sont souvent meilleurs que *jitter* lorsque l'on veut un rendu « naturel » ou « analogique » de sons numériques. On peut aussi les utiliser dans la composition algorithmique, pour générer des lignes mélodiques aléatoires lisses lors d'une utilisation conjointe avec l'opcode *samphold*.

Noter que le résultat est assez différent de celui que l'on obtiendrait en filtrant un bruit blanc, et que l'on peut ainsi obtenir un contrôle bien plus précis.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.15

# k

k — Convertit un paramètre de taux-i en une valeur de taux-k.

## Description

Convertit une valeur de taux-i en une valeur de taux-k, par exemple pour une utilisation avec *rnd()* et *birnd()* pour générer des nombres aléatoires au taux-k.

## Syntaxe

**k**(x) (arguments de taux-i seulement)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Voir Aussi

*i a*

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 5.00 de Csound

# kbetarand

kbetarand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.

# kbexprnd

kbexprnd — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.

# kcauchy

kcauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

# kdump

kdump — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk*.



# kdump2

kdump2 — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk2*.

# kdump3

kdump3 — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk3*.

# kdump4

kdump4 — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk4*.

# kexprand

kexprand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.

# kfilter2

kfilter2 — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *filter2*.

## Crédits

Auteur : Michael A. Casey  
M.I.T.  
Cambridge, Mass.  
1997

Nouveau dans la version 3.47

# kgauss

kgauss — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.

# kgoto

kgoto — Transfère le contrôle lors des phases d'exécution.

## Description

Transfère le contrôle sans condition vers l'instruction étiquetée par *label*, lors des phases d'exécution seulement.

## Syntaxe

```
kgoto label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.

## Exemples

Voici un exemple de l'opcode kgoto. Il utilise le fichier *kgoto.csd* [examples/kgoto.csd].

### Exemple 270. Exemple de l'opcode kgoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o kgoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Change kval linearly from 0 to 2 over
; the period set by the third p-field.
kval line 0, p3, 2

; If kval is greater than or equal to 1 then play the high note.
; If not then play the low note.
if (kval >= 1) kgoto highnote
kgoto lownote

highnote:
kfreq = 880
goto playit

lownote:
kfreq = 440
goto playit

playit:
; Print the values of kval and kfreq.
```

```
    printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq
    al oscil 10000, kfreq, 1
    out al
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
kval = 1.999639, kfreq = 880.000000
```

## Voir Aussi

*cggoto, cigoto, ckgoto, goto, if, igoto, tigoto, timeout*

## Crédits

Exemple écrit by Kevin Conder.



# klinrand

klinrand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.

# kon

kon — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *midion*.

# koutat

koutat — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkat*.

# koutc

koutc — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkc*.

# koutc14

koutc14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkc14*.

# koutpat

koutpat — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpat*.

# koutpb

koutpb — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpb*.

# koutpc

koutpc — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpc*.



# kpcauchy

kpcauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

# kpoisson

kpoisson — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

# kpow

kpow — Obsolète.

## Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

# kr

kr — Fixe le taux de contrôle.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

```
kr = iarg
```

## Initialisation

*kr* = (facultatif) -- fixe le taux de contrôle à *iarg* échantillons par seconde. La valeur par défaut est 1000.

De plus, toute *variable globale* [56] peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *kr*. Csound utilisera les valeurs par défaut, ou calculera *kr* à partir des valeurs définies de *ksmps* et *sr*. Habituellement, il est mieux de ne spécifier que *ksmps* et *sr* et de laisser csound calculer *kr*.

## Exemples

```
sr = 10000
kr = 500
ksmps = 20
gil = sr/2.
ga init 0
itranspose = octpch(.01)
```

## Voir Aussi

*ksmps*, *nchnls*, *sr*

# kread

kread — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk*.

# kread2

kread2 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk2*.

# kread3

kread3 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk3*.

# kread4

kread4 — Obsolète.

## Description

Obsolète depuis la version 3.52. Use the *readk4* opcode instead.



# ksmps

ksmps — Fixe le nombre d'échantillons dans une période de contrôle.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

**ksmps** = iarg

## Initialisation

*ksmps* = (facultatif) -- fixe le nombre d'échantillons dans une période de contrôle. Cette valeur doit être égale à *sr/kr*. La valeur par défaut est 10.

De plus, toute *variable globale* [56] peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *ksmps*. Csound essaiera de calculer la valeur omise à partir des valeurs spécifiées pour *sr* et *kr*, mais le résultat devra être un nombre entier.



### Avertissement

ksmps doit avoir une valeur entière.

## Exemples

```
sr = 10000
kr = 500
ksmps = 20
gil = sr/2.
ga init 0
itranspose = octpch(.01)
```

## Voir Aussi

*kr*, *nchnls*, *sr*

## Crédits

Grâce à une note de Gabriel Maldonado, un avertissement sur les valeurs entières a été ajouté.

# ktableseg

ktableseg — Deprecated.

## Description

Deprecated. Use the *tableseg* opcode instead.

## Syntax

```
ktableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

# ktrirand

ktrirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.

# kunirand

kunirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.

# kweibull

kweibull — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

# lfo

lfo — Un oscillateur basse fréquence avec différentes formes d'onde.

## Description

Un oscillateur basse fréquence avec différentes formes d'onde.

## Syntaxe

```
kres lfo kamp, kcps [, itype]
```

```
ares lfo kamp, kcps [, itype]
```

## Initialisation

*itype* (facultatif, par défaut 0) -- détermine la forme d'onde de l'oscillateur. La valeur par défaut est 0.

- *itype* = 0 - sinus
- *itype* = 1 - triangle
- *itype* = 2 - carrée (bipolaire)
- *itype* = 3 - carrée (unipolaire)
- *itype* = 4 - dent de scie
- *itype* = 5 - dent de scie (vers le bas)

L'onde sinus est implémentée comme une table de 4096 éléments avec interpolation linéaire. Les autres sont calculées.

## Exécution

*kamp* -- amplitude de la sortie

*kcps* -- fréquence de l'oscillateur

## Exemples

Voici un exemple de l'opcode lfo. Il utilise le fichier *lfo.csd* [examples/lfo.csd].

### Exemple 271. Exemple de l'opcode lfo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc          -d          ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o lfo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 10
  kcps = 5
  itype = 4

  k1 lfo kamp, kcps, itype
  ar oscil p4, p5+k1, 1
  out ar
endin

</CsInstruments>
<CsScore>

; Table #1: an ordinary sine wave.
f 1 0 32768 10 1

; p4 = amplitude of the output signal.
; p5 = frequency (in cycles per second) of the output signal.
; Play Instrument #1 for two seconds.
i 1 0 2 10000 220
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch  
 University of Bath/Codemist Ltd.  
 Bath, UK  
 Novembre 1998

Nouveau dans la version 3.491 de Csound

# limit

`limit` — Sets the lower and upper limits of the value it processes.

## Description

Sets the lower and upper limits of the value it processes.

## Syntax

```
ares limit asig, klow, khigh
```

```
ires limit isig, ilow, ihigh
```

```
kres limit ksig, klow, khigh
```

## Initialization

*isig* -- input signal

*ilow* -- low threshold

*ihigh* -- high threshold

## Performance

*xsig* -- input signal

*klow* -- low threshold

*khigh* -- high threshold

*limit* sets the lower and upper limits on the *xsig* value it processes. If *xhigh* is lower than *xlow*, then the output will be the average of the two - it will not be affected by *xsig*.

This opcode is useful in several situations, such as table indexing or for clipping and modeling a-rate, i-rate or k-rate signals.

## See Also

*mirror*, *wrap*

## Credits

Author: Robin Whittle  
Australia

New in Csound version 3.46



# line

line — Trace un segment de droite entre les points spécifiés.

## Description

Trace un segment de droite entre les points spécifiés.

## Syntaxe

```
ares line ia, idur, ib
```

```
kres line ia, idur, ib
```

## Initialisation

*ia* -- valeur initiale.

*ib* -- valeur après *idur* secondes.

*idur* -- durée en secondes du segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

## Exécution

*line* génère des signaux de contrôle ou des signaux audio dont les valeurs évoluent linéairement depuis une valeur initiale jusqu'à une valeur finale.



### Note

Une erreur habituelle avec cet opcode est de croire que la valeur de *ib* est tenue après la durée *idur*. *line* ne s'arrête pas automatiquement à la fin de la durée donnée. Si la longueur de votre note dépasse *idur* secondes, *kres* (ou *ares*) ne s'arrêtera pas sur *ib*, mais au contraire il continuera à monter ou à descendre à la même vitesse. Si l'on a besoin d'une pente ascendante (ou descendante) suivie d'une tenue il faut utiliser l'opcode *linseg*.

## Exemples

Voici un exemple de l'opcode *line*. Il utilise le fichier *line.csd* [examples/line.csd].

### Exemple 272. Exemple de l'opcode *line*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```

; -o line.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Define kcps as a frequency value that linearly declines
; from 880 to 220. It declines over the period set by p3.
kcps line 880, p3, 220

a1 oscil 20000, kcps, 1
out a1
endin

instr 2
kcps line 880, 1, 660 ; kcps won't stop at 660 if p3 > 1
a1 oscil 20000, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2

; Play Instrument #2 for two seconds.
i 2 3 2

e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*expon, expseg, expsegr, linseg, linsegr*

## Crédits

Exemple écrit par Kevin Conder.

# linen

*linen* — Applique un motif constitué d'une attaque et d'une chute en segments de droite à un signal d'amplitude.

## Description

*linen* -- applique un motif constitué d'une attaque et d'une chute en segments de droite à un signal d'amplitude.

## Syntaxe

```
ares linen xamp, irise, idur, idec
```

```
kres linen kamp, irise, idur, idec
```

## Initialisation

*irise* -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

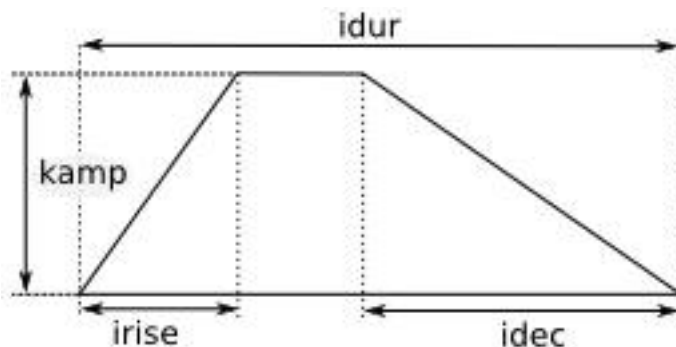
*idur* -- durée totale en secondes. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

*idec* -- durée de la chute en secondes. Si *idec* > *idur* la chute sera tronquée.

## Exécution

*kamp*, *xamp* -- signal d'amplitude en entrée.

L'attaque est appliquée pendant les *irise* premières secondes, et la chute à partir de *idur* - *idec*. Si ces périodes sont séparées dans le temps il y aura un entretien durant lequel *amp* ne sera pas modifié. Si l'attaque et la chute de *linen* se chevauchent, les deux modifications agiront en même temps pendant cette période. Si la durée totale *idur* est dépassée pendant l'exécution, la chute continuera dans la même direction, devenant négative.



Enveloppe générée par l'opcode *linen*



### Note

Il est faux de croire que la valeur 0 sera tenue après la fin de l'enveloppe à *idur* secondes. *linen* ne se termine pas automatiquement à la fin de la durée donnée. Si la longueur de la note est supérieure à *idur* secondes, *kres* (ou *ares*) ne s'arrêtera pas à 0, mais continuera au contraire à chuter à la même vitesse. Si l'on a besoin d'une chute suivie d'une valeur stable il vaut mieux utiliser l'opcode *linseg*.

## Voir Aussi

*envlpx*, *envlpxr*, *linenr*

# linenr

linenr — L'opcode *linen* rallongé avec un segment de relâchement.

## Description

*linenr* -- comme *linen* sauf que le dernier segment n'est entamé qu'après la détection d'un relâchement de note MIDI. La note est alors rallongée de la durée de la chute.

## Syntaxe

```
ares linenr xamp, irise, idec, iatdec
```

```
kres linenr kamp, irise, idec, iatdec
```

## Initialisation

*irise* -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

*idec* -- durée de la chute en secondes. Si *idec* > *idur* la chute sera tronquée.

*iatdec* -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulle ou négatives sont interdites.

## Exécution

*kamp*, *xamp* -- signal d'amplitude en entrée.

Ce qui rend unique *linenr* dans Csound c'est qu'il contient un *détecteur de note-off* et un *allongement de la durée de relâchement*. Lorsqu'il détecte la fin d'un évènement de partition ou un note-off MIDI, il allonge immédiatement la durée d'exécution de l'instrument courant de *idec* secondes, puis il exécute une chute exponentielle vers le facteur *iatdec*. S'il y a plusieurs unités dans un instrument, l'allongement est défini par le plus grand *idec*.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *linenr*, car la durée est allongée automatiquement.

Ces unités « r » peuvent être modifiées également par des évènements MIDI note-off provoqués par une vitesse nulle.

## Voir Aussi

*linsegr*, *expsegr*, *envlpxr*, *mxadsr*, *madsr*, *envlpx*, *linen*, *xtratim*

# lineto

lineto — Génère un glissando à partir d'un signal de contrôle.

## Description

Génère un glissando à partir d'un signal de contrôle.

## Syntaxe

```
kres lineto ksig, ktime
```

## Exécution

*kres* -- Signal de sortie.

*ksig* -- Signal d'entrée.

*ktime* -- Durée du glissando en secondes.

*lineto* ajoute un glissando (c-à-d des segments de droite) à un signal d'entrée en escalier (produit par exemple par *randh* ou par *lpshold*). Il génère un segment de droite allant d'un degré à l'autre en *ktime* secondes. Lorsque le degré suivant est atteint, cette valeur est maintenue jusqu'à ce qu'un nouveau degré apparaisse. Il faut s'assurer que la valeur de l'argument *ktime* est inférieure à l'intervalle de temps entre deux degrés consécutifs du signal original, sinon des discontinuités apparaîtront dans le signal de sortie.

Lorsqu'on l'utilise avec la sortie de *lpshold*, on obtient une simulation de l'effet de glissando des vieux synthétiseurs analogiques.



### Note

Une nouvelle valeur de *ksig* ou de *ktime* n'aura d'effet qu'après que la valeur précédente de *ktime* se soit écoulée.

## Voir Aussi

*tlineto*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

# linrand

linrand — Générateur de nombres aléatoires de distribution linéaire (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution linéaire (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares linrand krange
```

```
ires linrand krange
```

```
kres linrand krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (0 - *krange*). Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode linrand. Il utilise le fichier *linrand.csd* [examples/linrand.csd].

### Exemple 273. Exemple de l'opcode linrand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o linrand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Generate a random number between 0 and 1.
  ; krange = 1

  i1 linrand 1

  print i1
endin

; Instrument #2.
instr 2
  ; Generate a random number between 0 and 1.
  ; krange = 1

  seed 0

  i1 linrand 1

  print i1
endin

</CsInstruments>
<CsScore>
; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1: i1 = 0.394
```

## Voir Aussi

*seed, betarand, bexprnd, cauchy, exprand, gauss, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.



# linseg

linseg — Trace une suite de segments de droite entre les points spécifiés.

## Description

Trace une suite de segments de droite entre les points spécifiés.

## Syntaxe

```
ares linseg ia, idur1, ib [, idur2] [, ic] [...]
```

```
kres linseg ia, idur1, ib [, idur2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale.

*ib*, *ic*, etc. -- valeur après *dur1* secondes, etc.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.



### Note

Une erreur habituelle avec cet opcode est de croire que la dernière valeur est tenue après la durée totale. *linseg* ne s'arrête pas automatiquement à la fin de la durée totale. Si la longueur de votre note dépasse la somme de tous les *idur*, *kres* (ou *ares*) ne s'arrêtera pas sur la dernière valeur donnée, mais au contraire il continuera à monter ou à descendre à la même vitesse. On peut ajouter un segment final avec la même valeur que la précédente pour créer une valeur finale tenue.

## Exemples

Voici un exemple de l'opcode *linseg*. Il utilise le fichier *linseg.csd* [examples/linseg.csd].

**Exemple 274. Exemple de l'opcode *linseg*.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o linseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; p4 = frequency in pitch-class notation.
kcps = cpspch(p4)

; Create an amplitude envelope.
kenv linseg 0, p3*0.25, 1, p3*0.75, 0
kamp = kenv * 30000

a1 oscil kamp, kcps, 1
out a1
endin

instr 2
; p4 = frequency in pitch-class notation.
kcps = cpspch(p4)

; Create an amplitude envelope.
kenv linseg 0, 0.25, 1, 0.75, 0 ; kenv will go into negative if p3 > 1
kamp = kenv * 30000

a1 oscil kamp, kcps, 1
out a1
endin

instr 3
; p4 = frequency in pitch-class notation.
kcps = cpspch(p4)

; Create an amplitude envelope.
kenv linseg 0, 0.25, 1, 0.75, 0, 1, 0 ; kenv will stay at 0 indefinitely at the end
kamp = kenv * 30000

a1 oscil kamp, kcps, 1
out a1
endin
</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03

i 2 4 1.5 8.00 ; Notice the problem with linseg
i 3 6 1.5 8.00 ; this is the solution (instr 3)
e

</CsScore>
```

`</CsoundSynthesizer>`

## Voir Aussi

*expon, expseg, expsegr, line, linsegr transeg*

## Crédits

Exemple écrit par Kevin Conder.

# linsegr

linsegr — Trace une suite de segments de droite entre les points spécifiés avec un segment de relâchement.

## Description

Trace une suite de segments de droite entre les points spécifiés avec un segment de relâchement.

## Syntaxe

```
ares linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

```
kres linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

## Initialisation

*ia* -- valeur initiale.

*ib*, *ic*, etc. -- valeur après *dur1* secondes, etc.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

*irel*, *iz* -- durée en secondes et valeur finale du segment de relâchement de la note.

Pour les versions de Csound antérieures à la 5.00, le temps de relâchement ne peut pas dépasser 32767/*kr* secondes. Cette limite a été étendue à  $(2^{31}-1)/kr$ .

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

*linsegr* fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un événement ou MIDI noteoff est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *irel* secondes, de façon à ce que la valeur *iz* soit atteinte à la fin de cette période (quelque soit le segment dans lequel se trouvait l'unité). Les unités « r » peuvent aussi être modifiées par les vélocités nulles provoquant un message MIDI noteoff. S'il y a plusieurs extensions de durée dans un instrument, c'est la plus longue qui sera choisie.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linenr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter que qu'il n'est pas nécessaire d'utiliser *xtratim* avec *linsegr*, car la durée est allongée automatiquement.

## Exemples

Voici un exemple de l'opcode `linsegr`. Il utilise le fichier `linsegr.csd` [examples/linsegr.csd].

### Exemple 275. Exemple de l'opcode `linsegr`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o linsegr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  p4 = frequency in pitch-class notation.
  kcps = cpspch(p4)

  ; Use an amplitude envelope with second-long release.
  kenv linsegr 1, p3, 0.25, 1, 0
  kamp = kenv * 30000

  a1 oscil kamp, kcps, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Make sure the score lasts for four seconds.
f 0 4

; p4 = frequency (in pitch-class notation).
; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*linenr, expsegr, envlpxr, mxadsr, madsr expon, expsegr, expsega line, linsegr, xtratim, transegr*

## Crédits

Auteur : Barry L. Vercoe

Exemple écrit par Kevin Conder.

Décembre 2002, Décembre 2006. Merci à Istvan Varga pour l'ajout de la documentation sur le temps de relâchement maximum.

Nouveau dans Csound 3.47

# locsend

locsend — Distributes the audio signals of a previous *locsig* opcode.

## Description

*locsend* depends upon the existence of a previously defined *locsig*. The number of output signals must match the number in the previous *locsig*. The output signals from *locsend* are derived from the values given for distance and reverb in the *locsig* and are ready to be sent to local or global reverb units (see example below). The reverb amount and the balance between the 2 or 4 channels are calculated in the same way as described in the Dodge book (an essential text!).

## Syntax

```
a1, a2 locsend
```

```
a1, a2, a3, a4 locsend
```

## Examples

```
asig some audio signal
kdegree      line    0, p3, 360
kdistance     line    1, p3, 10
a1, a2, a3, a4 locsig asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4 locsend
ga1 = ga1+ar1
ga2 = ga2+ar2
ga3 = ga3+ar3
ga4 = ga4+ar4
                                outq    a1, a2, a3, a4
endin

instr 99 ; reverb instrument
a1      reverb2 ga1, 2.5, .5
a2      reverb2 ga2, 2.5, .5
a3      reverb2 ga3, 2.5, .5
a4      reverb2 ga4, 2.5, .5
                                outq    a1, a2, a3, a4

ga1=0
ga2=0
ga3=0
ga4=0
```

In the above example, the signal, *asig*, is sent around a complete circle once during the duration of a note while at the same time it becomes more and more « distant » from the listeners' location. *locsig* sends the appropriate amount of the signal internally to *locsend*. The outputs of the *locsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

*locsig* is useful for quad and stereo panning as well as fixed placed of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field.

```
instr 1
  al, a2          locsig asig, p4, p5, .1
  ar1, ar2        locsend
  ga1=ga1+ar1
  ga2=ga2+ar2
                  outs al, a
endin
instr 99
  ; reverb....
endin
```

A few notes:

```
;place the sound in the left speaker and near:
il 0 1 0 1

;place the sound in the right speaker and far:
il 1 1 90 25

;place the sound equally between left and right and in the middle ground distance:
il 2 1 45 12
e
```

The next example shows a simple intuitive use of the distance value to simulate Doppler shift. The same value is used to scale the frequency as is used as the distance input to *locsig*.

```
kdistance      line 1, p3, 10
kfreq = (ifreq * 340) / (340 + kdistance)
asig           oscili iamp, kfreq, 1
kdegree        line 0, p3, 360
al, a2, a3, a4 locsig asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4 locsend
```

## See Also

*locsig*

## Credits

Author: Richard Karpen  
Seattle, WA USA  
1998

New in Csound version 3.48



# locsig

locsig — Takes an input signal and distributes between 2 or 4 channels.

## Description

*locsig* takes an input signal and distributes it among 2 or 4 channels using values in degrees to calculate the balance between adjacent channels. It also takes arguments for distance (used to attenuate signals that are to sound as if they are some distance further than the loudspeaker itself), and for the amount the signal that will be sent to reverberators. This unit is based upon the example in the Charles Dodge/Thomas Jerse book, *Computer Music*, page 320.

## Syntax

```
a1, a2 locsig asig, kdegree, kdistance, kreverbsend
```

```
a1, a2, a3, a4 locsig asig, kdegree, kdistance, kreverbsend
```

## Performance

*kdegree* -- value between 0 and 360 for placement of the signal in a 2 or 4 channel space configured as: a1=0, a2=90, a3=180, a4=270 (kdegree=45 would balanced the signal equally between a1 and a2). *locsig* maps *kdegree* to sin and cos functions to derive the signal balances (ie.: asig=1, kdegree=45, a1=a2=.707).

*kdistance* -- value  $\geq 1$  used to attenuate the signal and to calculate reverb level to simulate distance cues. As *kdistance* gets larger the sound should get softer and somewhat more reverberant (assuming the use of *locsend* in this case).

*kreverbsend* -- the percentage of the direct signal that will be factored along with the distance and degree values to derive signal amounts that can be sent to a reverb unit such as reverb, or reverb2.

## Examples

```
asig some audio signal
kdegree      line    0, p3, 360
kdistance    line    1, p3, 10
a1, a2, a3, a4 locsig asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4 locsend
ga1 = ga1+ar1
ga2 = ga2+ar2
ga3 = ga3+ar3
ga4 = ga4+ar4
                                outq    a1, a2, a3, a4
endin

instr 99 ; reverb instrument
a1      reverb2 ga1, 2.5, .5
a2      reverb2 ga2, 2.5, .5
a3      reverb2 ga3, 2.5, .5
a4      reverb2 ga4, 2.5, .5
                                outq    a1, a2, a3, a4

ga1=0
ga2=0
ga3=0
ga4=0
```

In the above example, the signal, *asig*, is sent around a complete circle once during the duration of a note while at the same time it becomes more and more "distant" from the listeners' location. *locsig* sends the appropriate amount of the signal internally to *locsend*. The outputs of the *locsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

*locsig* is useful for quad and stereo panning as well as fixed placed of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field.

```
instr 1
  a1, a2          locsig asig, p4, p5, .1
  ar1, ar2        locsend
  ga1=ga1+ar1
  ga2=ga2+ar2
                  outs a1, a
endin
instr 99
  ; reverb....
endin
```

A few notes:

```
;place the sound in the left speaker and near:
il 0 1 0 1

;place the sound in the right speaker and far:
il 1 1 90 25

;place the sound equally between left and right and in the middle ground distance:
il 2 1 45 12
e
```

The next example shows a simple intuitive use of the distance value to simulate Doppler shift. The same value is used to scale the frequency as is used as the distance input to *locsig*.

```
kdistance      line      1, p3, 10
kfreq = (ifreq * 340) / (340 + kdistance)
asig           oscili    iamp, kfreq, 1
kdegree        line      0, p3, 360
a1, a2, a3, a4  locsig    asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4 locsend
```

## See Also

*locsend*

## Credits

Author: Richard Karpen  
Seattle, WA USA  
1998

New in Csound version 3.48

# log

log — Retourne un logarithme naturel.

## Description

Retourne le logarithme naturel de  $x$  ( $x$  strictement positif).

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

## Syntaxe

`log(x)` (pas de restriction de taux)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode *log*. Il utilise le fichier *log.csd* [examples/log.csd].

### Exemple 276. Exemple de l'opcode log.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o log.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = log(8)
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i1 = 2.079
```

## Voir Aussi

*abs, exp, frac, int, log10, i, sqrt*

## Crédits

Ecrit par John ffitch.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# log10

log10 — Retourne un logarithme en base 10.

## Description

Retourne le logarithme en base 10 de  $x$  ( $x$  strictement positif).

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

## Syntaxe

`log10(x)` (pas de restriction de taux)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode log10. Il utilise le fichier *log10.csd* [examples/log10.csd].

### Exemple 277. Exemple de l'opcode log10.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o log10.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = log10(8)
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i1 = 0.903
```

## Voir Aussi

*abs, exp, frac, int, log, i, sqrt*

## Crédits

Ecrit par John ffitch.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# logbtwo

logbtwo — Calcule le logarithme en base deux.

## Description

Calcule le logarithme en base deux.

## Syntaxe

`logbtwo(x)` (argument au taux d'initialisation ou de contrôle seulement)

## Exécution

`logbtwo()` retourne le logarithme en base deux de  $x$ . L'intervalle des valeurs permises en argument va de 0,25 à 4 (c-à-d une réponse comprise entre -2 et +2 octaves). Cette fonction est l'inverse de `powoftwo()`.

Ces fonctions sont rapides, car elles lisent des valeurs stockées dans des tables. Elles sont très utiles lorsque l'on travaille avec des rapports de hauteurs. Elles travaillent au taux-i et au taux-k.

## Exemples

Voici un exemple de l'opcode `logbtwo`. Il utilise le fichier `logbtwo.csd` [examples/logbtwo.csd].

### Exemple 278. Exemple de l'opcode `logbtwo`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o logbtwo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = logbtwo(3)
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e
```



```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.585
```

## Voir Aussi

*powoftwo*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Juin 1998

Auteur : John ffitch  
Université de Bath, Codemist, Ltd.  
Bath, UK  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# logcurve

logcurve — Cet opcode implémente une formule qui génère une courbe logarithmique normalisée dans l'intervalle 0 - 1. Il est basé sur le travail dans Max / MSP de Eric Singer (c) 1994.

## Description

Génère une courbe logarithmique dans l'intervalle de 0 à 1 avec une raideur de pente arbitraire. Une raideur de pente inférieure ou égale à 1,0 lévera des erreurs NaN (Not-a-Number) et provoquera un comportement instable.

La formule utilisée pour le calcul de la courbe est :

$$\log(x * (y-1)+1) / (\log(y))$$

où x est égal à *kindex* et y est égal à *ksteepness*.

## Syntaxe

```
kout logcurve kindex, ksteepness
```

## Exécution

*kindex* -- Valeur d'indice. Attendue dans l'intervalle de 0 à 1.

*ksteepness* -- Raideur de la courbe générée. Avec des valeurs proches de 1,0 on obtient une courbe plus rectiligne alors qu'avec des valeurs plus grandes la courbe est plus raide.

*kout* -- Sortie pondérée.

## Exemples

Voici un exemple de l'opcode logcurve. Il utilise le fichier *logcurve.csd* [examples/logcurve.csd].

### Exemple 279. Exemple de l'opcode logcurve.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent
-odac        -iadc     -d      ;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 100
nchnls = 2

instr 1 ; logcurve test

kmod phasor 1/p3

kout logcurve kmod, p4
```

```
    printks "kmod = %f  kout = %f\\n", 0.1, kmod, kout
  endin

</CsInstruments>
<CsScore>

i1 0 10 2
i1 10 10 30
i1 20 10 0.5

e
</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*scale, gainslider, expcurve*

## Crédits

Auteur : David Akbari  
Octobre  
2006

# loop\_ge

loop\_ge — Constructions de boucle.

## Description

Construction d'opérations de boucle.

## Syntaxe

```
loop_ge    indx, idecr, imin, label
```

```
loop_ge    kndx, kdecr, kmin, label
```

## Initialisation

*indx* -- variable de taux-i, compteur de la boucle.

*idecr* -- valeur de décrémentation de la boucle.

*imin* -- valeur minimale de l'index de la boucle.

## Exécution

*kndx* -- variable de taux-k, compteur de la boucle.

*kdecr* -- valeur de décrémentation de la boucle.

*kmin* -- valeur minimale de l'index de la boucle.

L'action de **loop\_ge** est équivalente à

```
indx = indx - idecr
if (indx >= imin) igoto label
```

ou à

```
kndx = kndx - kdecr
if (kndx >= kmin) kgoto label
```

## Voir Aussi

*loop\_gt*, *loop\_le* et *loop\_lt*.

## Crédits

Istvan Varga. 2006

Nouveau dans la version 5.01 de Csound

# loop\_gt

loop\_gt — Constructions de boucle..

## Description

Construction d'opérations de boucle.

## Syntaxe

```
loop_gt    indx, idecr, imin, label
```

```
loop_gt    kndx, kdecr, kmin, label
```

## Initialisation

*indx* -- variable de taux-i, compteur de la boucle.

*idecr* -- valeur de décrémentation de la boucle.

*imin* -- valeur minimale de l'index de la boucle.

## Exécution

*kndx* -- variable de taux-k, compteur de la boucle.

*kdecr* -- valeur de décrémentation de la boucle.

*kmin* -- valeur minimale de l'index de la boucle.

L'action de **loop\_gt** est équivalente à

```
indx = indx - idecr
if (indx > imin) igoto label
```

ou à

```
kndx = kndx - kdecr
if (kndx > kmin) kgoto label
```

## Voir Aussi

*loop\_ge*, *loop\_le* et *loop\_lt*.

## Crédits

Istvan Varga.

Nouveau dans la version 5.01 de Csound

# loop\_le

loop\_le — Constructions de boucle.

## Description

Construction d'opérations de boucle.

## Syntaxe

```
loop_le    indx, incr, imax, label
```

```
loop_le    kndx, kncr, kmax, label
```

## Initialisation

*indx* -- variable de taux-i, compteur de la boucle.

*incr* -- valeur d'incrément de la boucle.

*imax* -- valeur maximale de l'index de la boucle.

## Exécution

*kndx* -- variable de taux-k, compteur de la boucle.

*knrc* -- valeur d'incrément de la boucle.

*kmax* -- valeur maximale de l'index de la boucle.

L'action de **loop\_le** est équivalente à

```
indx  = indx + incr
if (indx <= imax) igoto label
```

ou à

```
kndx  = kndx + knrc
if (kndx <= kmax) kgoto label
```

## Voir Aussi

*loop\_ge*, *loop\_gt* et *loop\_lt*.

## Crédits

Istvan Varga.

Nouveau dans la version 5.01 de Csound

# loop\_lt

loop\_lt — Constructions de boucle.

## Description

Construction d'opérations de boucle.

## Syntaxe

```
loop_lt    indx, incr, imax, label
```

```
loop_lt    kndx, kncr, kmax, label
```

## Initialisation

*indx* -- variable de taux-i, compteur de la boucle.

*incr* -- valeur d'incrément de la boucle.

*imax* -- valeur maximale de l'index de la boucle.

## Exécution

*kndx* -- variable de taux-k, compteur de la boucle.

*knrc* -- valeur d'incrément de la boucle.

*kmax* -- valeur maximale de l'index de la boucle.

L'action de **loop\_lt** est équivalente à

```
indx = indx + incr  
if (indx < imax) igoto label
```

ou à

```
kndx = kndx + knrc  
if (kndx < kmax) kgoto label
```

## Voir Aussi

*loop\_ge*, *loop\_gt* et *loop\_le*.

## Crédits

Istvan Varga.

Nouveau dans la version 5.01 de Csound

# loopseg

loopseg — Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés.

## Description

Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

## Syntaxe

```
ksig loopseg kfreq, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]
```

## Exécution

*ksig* -- Signal de sortie.

*kfreq* -- Taux de répétition en Hz ou en fraction de Hz.

*ktrig* -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

*ktime0...ktimeN* -- Dates des points ; exprimées en fraction d'une période.

*kvalue0...kvalueN* -- Valeurs des points.

L'opcode *loopseg* est semblable à *linseg*, mais l'enveloppe entière est parcourue en boucle au taux *kfreq*. Noter que les valeurs temporelles ne sont pas exprimées en secondes mais en fractions d'une période. Concrètement chaque durée est proportionnelle aux autres, et la durée du cycle complet est proportionnelle à la somme de toutes les valeurs de durée.

La somme de toutes les durées est ensuite pondérée en fonction de l'argument *kfreq*. Par exemple, considérant une enveloppe faite de 3 segments, chaque segment ayant une valeur de durée de 100, leur somme sera 300. Cette valeur représente la durée totale de l'enveloppe, et elle est divisée en 3 parties égales, une partie pour chaque segment.

Concrètement, la durée réelle de l'enveloppe en secondes est déterminée par *kfreq*. Si l'enveloppe est à nouveau constituée de 3 segments, mais cette fois-ci le premier et le dernier segments ayant une durée de 50, tandis que le segment central a une durée de 100, leur somme sera 200. Ici 200 représente la durée totale des 3 segments, et ainsi le segment central sera deux fois plus long que les autres segments.

Tous les paramètres peuvent varier au taux-k. Si les valeurs de fréquence sont négatives, l'enveloppe est lue à l'envers. *ktime0* doit toujours valoir 0, sauf si l'on désire un effet spécial.

## Exemples

Voici un exemple de l'opcode *loopseg*. Il utilise le fichier *loopseg.csd* [examples/loopseg.csd].

### Exemple 280. Exemple de l'opcode *loopseg*.



Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o loopseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1
instr 1
  kfreq line 1, p3, 20

  klp loopseg kfreq, 0, 0, 0, 0.5, 30000, 1, 0

  a1 oscil klp, 440, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for five seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*lpshold loopxseg*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

# loopsegg

loopsegg — Signaux de contrôle basés sur des segments de droite.

## Description

Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés. L'enveloppe entière peut être parcourue en boucle à une vitesse variable. Chaque coordonnée de segment peut aussi varier au taux-k.

## Syntaxe

```
ksig loopsegg kphase, kvalue0, kdur0, kvalue1 \  
[, kdur1, ... , kdurN-1, kvalueN]
```

## Exécution

*ksig* - signal de sortie.

*kphase* - point de la séquence lu, exprimé en fraction d'un cycle (de 0 à 1)

*kvalue0 ...kvalueN* - valeurs des points.

*kdur0 ...kdurN-1* - durée des points exprimée en fractions d'un cycle.

L'opcode *loopsegg* est semblable à *loopseg* ; la seule différence étant que, à la place de la fréquence, une phase variable est utilisée. Si l'on utilise un *phaseur* pour obtenir la valeur de la phase, on aura un comportement identique à celui de *loopseg*, mais on peut obtenir des résultats intéressants avec des phases à l'évolution non linéaire, ce qui rend *loopsegg* plus puissant et plus général que *loopseg*.

## Exemples

Voici un exemple de l'opcode *loopsegg*. Il utilise le fichier *loopsegg.csd* [examples/loopsegg.csd].

### Exemple 281. Exemple de l'opcode *loopsegg*.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  
-odac      -iadc      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o loopsegg.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
sr=44100  
ksmps=1  
nchnls=2  
  
; By Mark Van Peteghem 2008  
  
instr 1  
iphase = p4  
  
kenv    linen 1, 0.1, p3, 0.1  
kph_amp phasor 2, 0
```

```
kamp      loopsegg kph_amp, 60, 1, 30, 1, 60
kamp      = ampdb(kamp)*kenv

kph_freq  phasor 2, iphase
klow_freq line 200, p3, 100
kfreq     loopsegg kph_freq, 400, 1, klow_freq, 1, 400

asig      vco2 kamp, kfreq, 2, 0.5

          outs asig, asig

        endin

</CsInstruments>
<CsScore>
il 0 3 0
il + . 0.50
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5. (Auparavant, disponible seulement dans CsoundAV)

# looptseg

**looptseg** — Génère un signal de contrôle constitué de segments linéaires ou exponentiels délimités par deux ou plus points spécifiés.

## Description

Génère un signal de contrôle constitué de segments linéaires ou exponentiels contrôlables délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

## Syntaxe

```
ksig looptseg kfreq, ktrig, ktime0, kvalue0, ktype, [, ktime1] [,ktype1] [, kvalue1] \
    [, ktime2] [,ktype2] [, kvalue2] [...]
```

## Exécution

*ksig* -- Signal de sortie.

*kfreq* -- Taux de répétition en Hz ou en fraction de Hz.

*ktrig* -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

*ktime0...ktimeN* -- Dates des points ; exprimées en fraction d'une période.

*kvalue0...kvalueN* -- Valeurs des points.

*ktype0...ktypeN* -- forme de l'enveloppe. Si la valeur est 0, la forme est linéaire ; sinon c'est une exponentielle concave (type positif) ou une exponentielle convexe (type négatif).

L'opcode *looptseg* est semblable à *transeg*, mais l'enveloppe entière est parcourue en boucle au taux *kfreq*. Noter que les valeurs temporelles ne sont pas exprimées en secondes mais en fractions d'une période. Concrètement chaque durée est proportionnelle aux autres, et la durée du cycle complet est proportionnelle à la somme de toutes les valeurs de durée.

La somme de toutes les durées est ensuite pondérée en fonction de l'argument *kfreq*. Par exemple, considérant une enveloppe faite de 3 segments, chaque segment ayant une valeur de durée de 100, leur somme sera 300. Cette valeur représente la durée totale de l'enveloppe, et elle est divisée en 3 parties égales, une partie pour chaque segment.

Concrètement, la durée réelle de l'enveloppe en secondes est déterminée par *kfreq*. Si l'enveloppe est à nouveau constituée de 3 segments, mais cette fois-ci le premier et le dernier segments ayant une durée de 50, tandis que le segment central a une durée de 100, leur somme sera 200. Ici 200 représente la durée totale des 3 segments, et ainsi le segment central sera deux fois plus long que les autres segments.

Tous les paramètres peuvent varier au taux-k. Si les valeurs de fréquence sont négatives, l'enveloppe est lue à l'envers. *ktime0* doit toujours valoir 0, sauf si l'on désire un effet spécial.

## Exemples

Voici un exemple de l'opcode *looptseg*. Il utilise le fichier *looptseg.csd* [examples/looptseg.csd].

## Exemple 282. Exemple de l'opcode looptseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o looptseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1
instr 1
  kfreq line 1, p3, 20

  klp looptseg kfreq, 0, 0,      0, 1, 0.5,      30000, -1, 1,      0

  a1 oscil klp, 440, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for five seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*lpshold loopseg*

## Crédits

Auteur : John ffitch

Nouveau dans la version 5.12

# loopxseg

loopxseg — Génère un signal de contrôle constitué de segments exponentiels délimités par deux ou plus points spécifiés.

## Description

Génère un signal de contrôle constitué de segments exponentiels délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

## Syntaxe

```
ksig loopxseg kfreq, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]
```

## Exécution

*ksig* -- Signal de sortie.

*kfreq* -- Taux de répétition en Hz ou en fraction de Hz.

*ktrig* -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

*ktime0...ktimeN* -- Dates des points ; exprimées en fraction d'une période.

*kvalue0...kvalueN* -- Valeurs des points.

L'opcode *loopxseg* est semblable à *expseg*, mais l'enveloppe entière est parcourue en boucle au taux *kfreq*. Noter que les valeurs temporelles ne sont pas exprimées en secondes mais en fractions d'une période. Concrètement chaque durée est proportionnelle aux autres, et la durée du cycle complet est proportionnelle à la somme de toutes les valeurs de durée.

La somme de toutes les durées est ensuite pondérée en fonction de l'argument *kfreq*. Par exemple, considérant une enveloppe faite de 3 segments, chaque segment ayant une valeur de durée de 100, leur somme sera 300. Cette valeur représente la durée totale de l'enveloppe, et elle est divisée en 3 parties égales, une partie pour chaque segment.

Concrètement, la durée réelle de l'enveloppe en secondes est déterminée par *kfreq*. Si l'enveloppe est à nouveau constituée de 3 segments, mais cette fois-ci le premier et le dernier segments ayant une durée de 50, tandis que le segment central a une durée de 100, leur somme sera 200. Ici 200 représente la durée totale des 3 segments, et ainsi le segment central sera deux fois plus long que les autres segments.

Tous les paramètres peuvent varier au taux-k. Si les valeurs de fréquence sont négatives, l'enveloppe est lue à l'envers. *ktime0* doit toujours valoir 0, sauf si l'on désire un effet spécial.

## Exemples

Voici un exemple de l'opcode *loopxseg*. Il utilise le fichier *loopxseg.csd* [examples/loopxseg.csd].

### Exemple 283. Exemple de l'opcode *loopxseg*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o loopxseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1
instr 1
  kfreq line 1, p3, 20

  klp loopxseg kfreq, 0, 0, 0, 0.5, 30000, 1, 0

  a1 oscil klp, 440, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for five seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*lpshold loopseg*

## Crédits

Auteur : John ffitich

Nouveau dans la version 5.12

# lorenz

lorenz — Implémente le système d'équations de Lorenz.

## Description

Implémente le système d'équations de Lorenz. Le système de Lorenz est un système dynamique chaotique qui fut utilisé à l'origine pour simuler le mouvement d'une particule dans des courants de convection et des systèmes météorologiques simplifiés. De petites différences dans les conditions initiales conduisent rapidement à des valeurs divergentes. C'est ce qu'on appelle parfois l'effet papillon. Si un papillon bat des ailes en Australie, cela aura des conséquences sur le temps en Alaska. Ce système est l'un des éléments fondateurs du développement de la théorie du chaos. Il est utile comme source audio chaotique ou comme source de modulation basse fréquence.

## Syntaxe

```
ax, ay, az lorenz ksv, krsv, kbv, kh, ix, iy, iz, iskip [, iskipinit]
```

## Initialisation

*ix, iy, iz* -- les coordonnées initiales de la particule.

*iskip* -- utilisé pour sauter des valeurs générées. Si *iskip* vaut 5, seulement une valeur sur cinq sera retournée. Utile pour générer des sons de hauteur plus élevée.

*iskipinit* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre sera ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

## Exécution

*ksv* -- le nombre de Prandtl ou sigma

*krv* -- le nombre de Rayleigh

*kbv* -- le rapport entre la longueur et la largeur de la boîte dans laquelle les courants de convection sont générés

*kh* -- le pas de progression utilisé pour le calcul approché de l'équation différentielle. On peut l'utiliser pour contrôler la hauteur du système. Des valeurs comprises entre 0,1 et 0,001 sont typiques.

Le calcul approché des équations se fait comme suit :

$$\begin{aligned}x &= x + h*(s*(y - x)) \\ y &= y + h*(-x*z + r*x - y) \\ z &= z + h*(x*y - b*z)\end{aligned}$$

Les valeurs historiques des paramètres sont :

ks = 10  
kr = 28



kb = 8/3



## Note

Cet algorithme utilise des boucles de rétroaction internes non linéaires ce qui fait dépendre le résultat audio du taux d'échantillonnage de l'orchestre. Par exemple, si l'on développe un projet avec  $sr=48000\text{Hz}$  et si l'on veut produire un CD audio de ce projet, il faut enregistrer un fichier avec  $sr=48000\text{Hz}$ , puis sous-échantillonner ce fichier à  $44100\text{Hz}$  avec l'utilitaire *srconv*.

## Exemples

Voici un exemple de l'opcode *lorenz*. Il utilise le fichier *lorenz.csd* [examples/lorenz.csd].

### Exemple 284. Exemple de l'opcode *lorenz*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lorenz.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1 - a lorenz system in 3D space.
instr 1
; Create a basic tone.
kamp init 25000
kcps init 220
ifn = 1
asnd oscil kamp, kcps, ifn

; Figure out its X, Y, Z coordinates.
ksv init 10
krv init 28
kbv init 2.667
kh init 0.0003
ix = 0.6
iy = 0.6
iz = 0.6
iskip = 1
axl, ayl, azl lorenz ksv, krv, kbv, kh, ix, iy, iz, iskip

; Place the basic tone within 3D space.
kx downsamp axl
ky downsamp ayl
kz downsamp azl
idist = 1
ift = 0
imode = 1
imdel = 1.018853416
iovr = 2
aw2, ax2, ay2, az2 spat3d asnd, kx, ky, kz, idist, \
                           ift, imode, imdel, iovr
```

```
; Convert the 3D sound to stereo.
aleft = aw2 + ay2
aright = aw2 - ay2

outs aleft, aright
endin

</CsInstruments>
<CsScore>

; Table #1 a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 5 seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Hans Mikelson  
Février 1999

Nouveau dans la version 3.53 de Csound

Note ajoutée par François Pinot, août 2009

# lorisread

**lorisread** — Imports a set of bandwidth-enhanced partials from a SDIF-format data file, applying control-rate frequency, amplitude, and bandwidth scaling envelopes, and stores the modified partials in memory.

## Syntax

```
lorisread ktmpnt, ifilcod, istoreidx, kfregenv, kampenv, kbwenv[, ifadetime]
```

## Description

**lorisread** imports a set of bandwidth-enhanced partials from a SDIF-format data file, applying control-rate frequency, amplitude, and bandwidth scaling envelopes, and stores the modified partials in memory.

## Initialization

*ifilcod* - integer or character-string denoting a control-file derived from reassigned bandwidth-enhanced analysis of an audio signal. An integer denotes the suffix of a file *loris.sdif* (e.g. *loris.sdif.1*); a character-string (in double quotes) gives a filename, optionally a full pathname. If not a full pathname, the file is sought first in the current directory, then in the one given by the environment variable *SADIR* (if defined). The reassigned bandwidth-enhanced data file contains breakpoint frequency, amplitude, noisiness, and phase envelope values organized for bandwidth-enhanced additive resynthesis. The control data must conform to one of the SDIF formats that can be

Loris stores partials in SDIF RBEP frames. Each RBEP frame contains one RBEP matrix, and each row in a RBEP matrix describes one breakpoint in a Loris partial. A RBEL frame containing one RBEL matrix describing the labeling of the partials may precede the first RBEP frame in the SDIF file. The RBEP and RBEL frame and matrix definitions are included in the SDIF file's header. In addition to RBEP frames, Loris can also read and write SDIF 1TRC frames. Since 1TRC frames do not represent bandwidth-enhancement or the exact timing of Loris breakpoints, their use is not recommended. 1TRC capabilities are provided to allow interchange with programs that are unable to handle RBEP frames.

*istoreidx*, *ireadidx*, *isrcidx*, *itgtidx* are labels that identify a stored set of bandwidth-enhanced partials. **lorisread** imports partials from a SDIF file and stores them with the integer label *istoreidx*. **lorismorph** morphs sets of partials labeled *isrcidx* and *itgtidx*, and stores the resulting partials with the integer label *istoreidx*. **lorisplay** renders the partials stored with the label *ireadidx*. The labels are used only at initialization time, and may be reused without any cost or benefit in efficiency, and without introducing any interaction between instruments or instances.

*ifadetime* (optional) - In general, partials exported from Loris begin and end at non-zero amplitude. In order to prevent artifacts, it is very often necessary to fade the partials in and out, instead of turning them abruptly on and off. Specification of a non-zero *ifadetime* causes partials to fade in at their onsets and to fade out at their terminations. This is achieved by adding two more breakpoints to each partial: one *ifadetime* seconds before the start time and another *ifadetime* seconds after the end time. (However, no breakpoint will be introduced at a time less than zero. If necessary, the onset fade time will be shortened.) The additional breakpoints at the partial onset and termination will have the same frequency and bandwidth as the first and last breakpoints in the partial, respectively, but their amplitudes will be zero. The phase of the new breakpoints will be extrapolated to preserve phase correctness. If no value is specified, *ifadetime* defaults to zero. Note that the *fadetime* may not be exact, since the partial parameter envelopes are sampled at the control rate (*krate*) and indexed by *ktmpnt* (see below), and not by real time.

## Performance

lorisread reads pre-computed Reassigned Bandwidth-Enhanced analysis data from a file stored in SDIF format, as described above. The passage of time through this file is specified by *ktimpnt*, which represents the time in seconds. *ktimpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file. *kfreqenv* is a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave. *kampenv* is a control-rate scale factor that is applied to all partial amplitude envelopes. *kbwenv* is a control-rate scale factor that is applied to all partial bandwidth or noisiness envelopes. The bandwidth-enhanced partial data is stored in memory with a specified label for future access by another generator.

## Credits

This implementation of the Loris unit generators was written by Kelly Fitz ([loris@cerlsoundgroup.org](mailto:loris@cerlsoundgroup.org) [<mailto:loris@cerlsoundgroup.org>]). It is patterned after a prototype implementation of the *lorisplay* unit generator written by Corbin Champion, and based on the method of Bandwidth-Enhanced Additive Synthesis and on the sound morphing algorithms implemented in the Loris library for sound modeling and manipulation. The opcodes were further adapted as a plugin for Csound 5 by Michael Gogins.

# lorismorph

**lorismorph** — Morphs two stored sets of bandwidth-enhanced partials and stores a new set of partials representing the morphed sound. The morph is performed by linearly interpolating the parameter envelopes (frequency, amplitude, and bandwidth, or noisiness) of the bandwidth-enhanced partials according to control-rate frequency, amplitude, and bandwidth morphing functions.

## Syntax

```
lorismorph isrcidx, itgtidx, istoreidx, kfreqmorphenv, kampmorphenv, kbwmorphenv
```

## Description

*lorismorph* morphs two stored sets of bandwidth-enhanced partials and stores a new set of partials representing the morphed sound. The morph is performed by linearly interpolating the parameter envelopes (frequency, amplitude, and bandwidth, or noisiness) of the bandwidth-enhanced partials according to control-rate frequency, amplitude, and bandwidth morphing functions.

## Initialization

*istoreidx*, *ireadidx*, *isrcidx*, *itgtidx* are labels that identify a stored set of bandwidth-enhanced partials. *lorisread* imports partials from a SDIF file and stores them with the integer label *istoreidx*. *lorismorph* morphs sets of partials labeled *isrcidx* and *itgtidx*, and stores the resulting partials with the integer label *istoreidx*. *lorisplay* renders the partials stored with the label *ireadidx*. The labels are used only at initialization time, and may be reused without any cost or benefit in efficiency, and without introducing any interaction between instruments or instances.

## Performance

*lorismorph* generates a set of bandwidth-enhanced partials by morphing two stored sets of partials, the source and target partials, which may have been imported using *lorisread*, or generated by another unit generator, including another instance of *lorismorph*. The morph is performed by interpolating the parameters of corresponding (labeled) partials in the two source sounds. The sound morph is described by three control-rate morphing envelopes. *kfreqmorphenv* describes the interpolation of partial frequency values in the two source sounds. When *kfreqmorphenv* is 0, partial frequencies are obtained from the partials stored at *isrcidx*. When *kfreqmorphenv* is 1, partial frequencies are obtained from the partials at *itgtidx*. When *kfreqmorphenv* is between 0 and 1, the partial frequencies are interpolated between corresponding source and target partials. Interpolation of partial amplitudes and bandwidth (noisiness) coefficients are similarly described by *kampmorphenv* and *kbwmorphenv*.

## Credits

This implementation of the Loris unit generators was written by Kelly Fitz (loris@cerlsoundgroup.org [mailto:loris@cerlsoundgroup.org]). It is patterned after a prototype implementation of the *lorisplay* unit generator written by Corbin Champion, and based on the method of Bandwidth-Enhanced Additive Synthesis and on the sound morphing algorithms implemented in the Loris library for sound modeling and manipulation. The opcodes were further adapted as a plugin for Csound 5 by Michael gogins.

# lorisplay

*lorisplay* — renders a stored set of bandwidth-enhanced partials using the method of Bandwidth-Enhanced Additive Synthesis implemented in the Loris software, applying control-rate frequency, amplitude, and bandwidth scaling envelopes.

## Syntax

```
ar lorisplay ireadidx, kfreqenv, kampenv, kbwenv
```

## Description

*lorisplay* renders a stored set of bandwidth-enhanced partials using the method of Bandwidth-Enhanced Additive Synthesis implemented in the Loris software, applying control-rate frequency, amplitude, and bandwidth scaling envelopes.

## Initialization

*istoreidx*, *ireadidx*, *isrcidx*, *itgtidx* are labels that identify a stored set of bandwidth-enhanced partials. *lorisread* imports partials from a SDIF file and stores them with the integer label *istoreidx*. *lorismorph* morphs sets of partials labeled *isrcidx* and *itgtidx*, and stores the resulting partials with the integer label *istoreidx*. *lorisplay* renders the partials stored with the label *ireadidx*. The labels are used only at initialization time, and may be reused without any cost or benefit in efficiency, and without introducing any interaction between instruments or instances.

## Performance

*lorisplay* implements signal reconstruction using Bandwidth-Enhanced Additive Synthesis. The control data is obtained from a stored set of bandwidth-enhanced partials imported from an SDIF file using *lorisread* or constructed by another unit generator such as *lorismorph*. *kfreqenv* is a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave. *kampenv* is a control-rate scale factor that is applied to all partial amplitude envelopes. *kbwenv* is a control-rate scale factor that is applied to all partial bandwidth or noisiness envelopes. The bandwidth-enhanced partial data is stored in memory with a specified label for future access by another generator.

## Credits

This implementation of the Loris unit generators was written by Kelly Fitz ([loris@cerlsoundgroup.org](mailto:loris@cerlsoundgroup.org) [<mailto:loris@cerlsoundgroup.org>]). It is patterned after a prototype implementation of the *lorisplay* unit generator written by Corbin Champion, and based on the method of Bandwidth-Enhanced Additive Synthesis and on the sound morphing algorithms implemented in the Loris library for sound modeling and manipulation. The opcodes were further adapted as a plugin for Csound 5 by Michael Gogins.

# loscil

loscil — Lit un son échantillonné depuis une table.

## Description

Lit un son échantillonné (mono ou stéréo) depuis une table, avec des boucles facultatives d'entretien et de relâchement.

## Syntaxe

```
ar1 [,ar2] loscil xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \  
[, imod2] [, ibeg2] [, iend2]
```

## Initialisation

*ifn* -- numéro de table de fonction, contenant typiquement un son échantillonné avec des points de boucle précisés, remplie au moyen de *GEN01*. Le fichier source peut être mono ou stéréo.

*ibas* (facultatif) -- fréquence de base en Hz du son enregistré. Elle remplace éventuellement la fréquence donnée dans le fichier audio, mais devient nécessaire si le fichier n'en contient pas. La valeur par défaut est 261,626 Hz, c-à-d le do médian. (Nouveau dans Csound 4.03). Si la valeur est inconnue ou absente il faut utiliser 1 ici et dans *kcps*.

*imod1*, *imod2* (facultatif, -1 par défaut) -- modes d'interprétation des boucles d'entretien et de relâchement. Une valeur de 1 signifie une boucle normale, 2 signifie une boucle à l'endroit et à l'envers, 0 signifie pas de boucle. La valeur par défaut (-1) s'en remet au mode et aux points de boucle définis dans le fichier source. Il faut s'assurer de choisir un mode approprié si le fichier ne contient pas cette information.

*ibeg1*, *iend1*, *ibeg2*, *iend2* (facultatifs, dépendants de *mod1*, *mod2*) -- début et fin des boucles d'entretien et de relâchement. Ils sont mesurés en *trames d'échantillon* depuis le début du fichier, et auront ainsi la même valeur que le son soit mono ou stéréo. Si aucun point de boucle n'est spécifié et qu'un mode de boucle est donné (*imod1*, *imod2*, le fichier sera lu en boucle sur toute sa longueur.

## Exécution

*ar1*, *ar2* -- la sortie de taux audio. Il n'y a que *ar1* pour une sortie mono, alors qu'il y a *ar1* et *ar2* pour une sortie stéréo.

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- la fréquence du signal de sortie en Hz.

*loscil* parcourt la ftable audio à un taux déterminé par *kcps*, en multipliant le résultat par *xamp*. L'incrément de lecture pour *kcps* dépend de la fréquence de base de la table *ibas*, et il est automatiquement ajusté si le taux d'échantillonnage *sr* de l'orchestre diffère de celui auquel la source a été enregistrée. Dans cette unité, ftable est toujours lue avec interpolation.

Si la lecture atteint la fin de la *boucle d'entretien* et que la boucle est active, le point de lecture sera modifié et *loscil* continuera sa lecture depuis l'intérieur de la boucle. Une fois que l'instrument reçoit un signal *turnoff* (depuis la partition ou depuis un événement MIDI noteoff), la fin de la boucle est ignorée et la lecture continue vers la fin de la *boucle de relâchement*, ou vers le dernier échantillon (dorénavant vers zéro).

*loscil* est l'unité de base pour bâtir un échantillonneur. Avec un ensemble suffisamment conséquent de sons de piano échantillonnés, par exemple, cette unité peut les rééchantillonner pour simuler les hauteurs manquantes. La détection de la source de son la plus proche d'une hauteur donnée peut être réalisée par la consultation d'une table. Une fois qu'un instrument échantillonneur est actif, son point de *turnoff* peut être imprévisible et nécessiter une enveloppe de *relâchement* externe ; on réalise souvent cela en munissant le son échantillonné d'un détecteur *linenr*, qui allonge la durée d'un instrument à la fin de la note d'une durée spécifique car il implémente une chute.

Si l'on veut boucler sur tout le fichier, il faut spécifier un mode de boucle dans *imodl* et ne donner aucune valeur pour *ibeg* et *iend*.



## Note pour les utilisateurs de Windows

Les utilisateurs de Windows utilisent normalement l'antislash, « \ », lorsqu'ils écrivent les chemins de leurs fichiers. Par exemple, un utilisateur de Windows pourra utiliser le chemin « c:\music\samples\loop001.wav ». Ceci pose problème car les l'antislash est normalement utilisé pour spécifier des caractères spéciaux.

Pour écrire correctement ce chemin dans Csound, on peut :

- Soit *utiliser le slash* : c:/music/samples/loop001.wav
- Soit *utiliser le caractère spécial d'antislash*, « \\ » : c:\\music\\samples\\loop001.wav



## Note

Voici *loscil* en mono :

```
a1 loscil 10000, 1, 1, 1, 1
```

... et *loscil* en stéréo :

```
a1, a2 loscil 10000, 1, 1, 1, 1
```

## Exemples

Voici un exemple de l'opcode *loscil*. Il utilise le fichier *loscil.csd* [examples/loscil.csd], and *beats.wav* [examples/beats.wav].

### Exemple 285. Exemple de l'opcode *loscil*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
```



```

-odac          -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o loscil.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  ; If you don't know the frequency of your audio file,
  ; set both the kcps and ibas parameters equal to 1.
  kcps = 1
  ifn = 1
  ibas = 1

  a1 loscil kamp, kcps, ifn, ibas
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1: an audio file.
; Its table size is deferred,
; and format taken from the soundfile header.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 6 seconds.
; This will loop the audio file several times.
i 1 0 6
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*loscil3* et *GEN01*

## Crédits

La note au sujet de la différence mono/stéréo est due à Rasmus Ekman.

Exemple écrit par Kevin Conder.

# loscil3

loscil3 — Lit un son échantillonné depuis une table avec interpolation cubique.

## Description

Lit un son échantillonné (mono ou stéréo) depuis une table, avec des boucles facultatives d'entretien et de relâchement, et interpolation cubique.

## Syntax

```
ar1 [,ar2] loscil3 xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \  
[, imod2] [, ibeg2] [, iend2]
```

## Initialisation

*ifn* -- numéro de table de fonction, contenant typiquement un son échantillonné avec des points de boucle précisés, remplie au moyen de *GEN01*. Le fichier source peut être mono ou stéréo.

*ibas* (facultatif) -- fréquence de base en Hz du son enregistré. Elle remplace éventuellement la fréquence donnée dans le fichier audio, mais devient nécessaire si le fichier n'en contient pas. La valeur par défaut est 261,626 Hz, c-à-d le do médian. (Nouveau dans Csound 4.03). Si la valeur est inconnue ou absente il faut utiliser 1 ici et dans *kcps*.

*imod1*, *imod2* (facultatif, -1 par défaut) -- modes d'interprétation des boucles d'entretien et de relâchement. Une valeur de 1 signifie une boucle normale, 2 signifie une boucle à l'endroit et à l'envers, 0 signifie pas de boucle. La valeur par défaut (-1) s'en remet au mode et aux points de boucle définis dans le fichier source. Il faut s'assurer de choisir un mode approprié si le fichier ne contient pas cette information.

*ibeg1*, *iend1*, *ibeg2*, *iend2* (facultatifs, dépendants de *mod1*, *mod2*) -- début et fin des boucles d'entretien et de relâchement. Ils sont mesurés en *trames d'échantillon* depuis le début du fichier, et auront ainsi la même valeur que le son soit mono ou stéréo. Si aucun point de boucle n'est spécifié et qu'un mode de boucle est donné (*imod1*, *imod2*, le fichier sera lu en boucle sur toute sa longueur.

## Exécution

*ar1*, *ar2* -- la sortie de taux audio. Il n'y a que *ar1* pour une sortie mono, alors qu'il y a *ar1* et *ar2* pour une sortie stéréo.

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- la fréquence du signal de sortie en Hz.

*loscil3* est identique à *loscil* sauf qu'il utilise l'interpolation cubique. Nouveau dans la version 3.50 de Csound.



### Note pour les utilisateurs de Windows

Les utilisateurs de Windows utilisent normalement l'antislash, « \ », lorsqu'ils écrivent les chemins de leurs fichiers. Par exemple, un utilisateur de Windows pourra utiliser le chemin « c:\music\samples\loop001.wav ». Ceci pose problème car les l'antislash est normalement utilisé pour spécifier des caractères spéciaux.

Pour écrire correctement ce chemin dans Csound, on peut :

- Soit *utiliser le slash* : c:/music/samples/loop001.wav
- Soit *utiliser le caractère spécial d'antislash*, « \\ »: c:\\music\\samples\\loop001.wav



## Note

Voici *loscil3* en mono :

```
a1 loscil3 10000, 1, 1, 1, 1
```

... et *loscil3* en stéréo :

```
a1, a2 loscil3 10000, 1, 1, 1, 1
```

## Exemples

Voici un exemple de l'opcode *loscil3*. Il utilise le fichier *loscil3.csd* [examples/loscil3.csd], and *beats.wav* [examples/beats.wav].

### Exemple 286. Exemple de l'opcode *loscil3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o loscil3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  ; If you don't know the frequency of your audio file,
  ; set both the kcps and ibas parameters equal to 1.
  kcps = 1
  ifn = 1
  ibas = 1

  a1 loscil3 kamp, kcps, ifn, ibas
  out a1
endin
```

```
</CsInstruments>
<CsScore>

; Table #1: an audio file.
; Its table size is deferred,
; and format taken from the soundfile header.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 6 seconds.
; This will loop the drum pattern several times.
i 1 0 6
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*loscil* et *GEN01*

## Crédits

La note au sujet de la différence mono/stéréo est due à Rasmus Ekman.

Exemple écrit par Kevin Conder.

# loscilx

loscilx — Oscillateur de boucle.

## Description

Cette notice reste à écrire, mais la syntaxe de l'opcode est correcte.

## Syntaxe

```
ar1 [, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
    ar15, ar16] loscilx xamp, kcps, ifn \
    [, iwsiz, ibas, istr, imod, ibeg, iend]
```

## Voir Aussi

*sndload*

*loscil*

## Crédits

Ecrit par Istvan Varga.

2006

Nouveau dans Csound 5.03

# lowpass2

lowpass2 — A resonant lowpass filter.

## Description

Implementation of a resonant second-order lowpass filter.

## Syntax

```
ares lowpass2 asig, kcf, kq [, iskip]
```

## Initialization

*iskip* -- initial disposition of internal data space. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*asig* -- input signal to be filtered

*kcf* -- cutoff or resonant frequency of the filter, measured in Hz

*kq* -- Q of the filter, defined, for bandpass filters, as bandwidth/cutoff. *kq* should be between 1 and 500

*lowpass2* is a second order IIR lowpass filter, with k-rate controls for cutoff frequency (*kcf*) and Q (*kq*). As *kq* is increased, a resonant peak forms around the cutoff frequency, transforming the lowpass filter response into a response that is similar to a bandpass filter, but with more low frequency energy. This corresponds to an increase in the magnitude and "sharpness" of the resonant peak. For high values of *kq*, a scaling function such as *balance* may be required. In practice, this allows for the simulation of the voltage-controlled filters of analog synthesizers, or for the creation of a pitch of constant amplitude while filtering white noise.

## Examples

Here is an example of the lowpass2 opcode. It uses the file *lowpass2.csd* [examples/lowpass2.csd].

### Exemple 287. Example of the lowpass2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lowpass2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Sean Costello */
```

```

; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
sr = 44100
kr = 2205
ksmps = 20
nchnls = 1

instr 1

idur   =      p3
ifreq  =      p4
iamp   =      p5 * .5
iharms =      (sr*.4) / ifreq

; Sawtooth-like waveform
asig   gbuzz 1, ifreq, iharms, 1, .9, 1

; Envelope to control filter cutoff
kfreq  linseg 1, idur * 0.5, 5000, idur * 0.5, 1

afilt  lowpass2 asig, kfreq, 30

; Simple amplitude envelope
kenv   linseg 0, .1, iamp, idur -.2, iamp, .1, 0
out    asig * kenv

endin

</CsInstruments>
<CsScore>

/* Written by Sean Costello */
f1 0 8192 9 1 1 .25

i1 0 5 100 1000
i1 5 5 200 1000
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Sean Costello  
 Seattle, Washington  
 August 1999

New in Csound version 4.0

# lowres

lowres — Another resonant lowpass filter.

## Description

*lowres* is a resonant lowpass filter.

## Syntax

```
ares lowres asig, kcutoff, kresonance [, iskip]
```

## Initialization

*iskip* -- initial disposition of internal data space. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*asig* -- input signal

*kcutoff* -- filter cutoff frequency point

*kresonance* -- resonance amount

*lowres* is a resonant lowpass filter derived from a Hans Mikelson orchestra. This implementation is much faster than implementing it in Csound language, and it allows *kr* lower than *sr*. *kcutoff* is not in Hz and *kresonance* is not in dB, so experiment for the finding best results.

## Examples

Here is an example of the *lowres* opcode. It uses the file *lowres.csd* [examples/lowres.csd] and *beats.wav* [examples/beats.wav].

### Exemple 288. Example of the *lowres* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lowres.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```



```
; Instrument #1.
instr 1
; Use a nice sawtooth waveform.
asig vco 5000, 440, 1

; Vary the cutoff frequency from 30 to 300 Hz.
kcutoff line 30, p3, 300
kresonance = 10

; Apply the filter.
al lowres asig, kcutoff, kresonance

out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*lowresx*

## Credits

Author: Gabriel Maldonado (adapted by John ffitich)  
Italy

Example written by Kevin Conder.

New in Csound version 3.49

# lowresx

lowresx — Simulates layers of serially connected resonant lowpass filters.

## Description

*lowresx* is equivalent to more layers of *lowres* with the same arguments serially connected.

## Syntax

```
ares lowresx asig, kcutoff, kresonance [, inumlayer] [, iskip]
```

## Initialization

*inumlayer* -- number of elements in a *lowresx* stack. Default value is 4. There is no maximum.

*iskip* -- initial disposition of internal data space. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*asig* -- input signal

*kcutoff* -- filter cutoff frequency point

*kresonance* -- resonance amount

*lowresx* is equivalent to more layer of *lowres* with the same arguments serially connected. Using a stack of a larger number of filters allows a sharper cutoff. This is faster than using a larger number of instances of *lowres* in a Csound orchestra because only one initialization and k cycle are needed at time and the audio loop falls entirely inside the cache memory of processor. Based on an orchestra by Hans Mielson

## Examples

Here is an example of the *lowresx* opcode. It uses the file *lowresx.csd* [examples/lowresx.csd], and *beats.wav* [examples/beats.wav].

### Exemple 289. Example of the lowresx opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lowresx.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - play the sawtooth waveform through a
; stack of filters.
instr 1
; Use a nice sawtooth waveform.
asig vco 5000, 440, 1

; Vary the cutoff frequency from 30 to 300 Hz.
kcutoff line 30, p3, 300
kresonance = 3
inumlayer = 2

alr lowresx asig, kcutoff, kresonance, inumlayer

; It gets loud, so clip the output amplitude to 30,000.
al clip alr, 1, 30000
out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## See Also

*lowres*

## Credits

Author: Gabriel Maldonado (adapted by John ffitich)  
Italy

New in Csound version 3.49

# lpf18

lpf18 — A 3-pole sweepable resonant lowpass filter.

## Description

Implementation of a 3 pole sweepable resonant lowpass filter.

## Syntax

```
ares lpf18 asig, kfco, kres, kdist
```

## Performance

*kfco* -- the filter cutoff frequency in Hz. Should be in the range 0 to  $sr/2$ .

*kres* -- the amount of resonance. Self-oscillation occurs when *kres* is approximately 1. Should usually be in the range 0 to 1, however, values slightly greater than 1 are possible for more sustained oscillation and an « overdrive » effect.

*kdist* -- amount of distortion. *kdist* = 0 gives a clean output. *kdist* > 0 adds *tanh()* distortion controlled by the filter parameters, in such a way that both low cutoff and high resonance increase the distortion amount. Some experimentation is encouraged.

*lpf18* is a digital emulation of a 3 pole (18 dB/oct.) lowpass filter capable of self-oscillation with a built-in distortion unit. It is really a 3-pole version of *moogvcf*, retuned, recalibrated and with some performance improvements. The tuning and feedback tables use no more than 6 adds and 6 multiplies per control rate. The distortion unit, itself, is based on a modified *tanh* function driven by the filter controls.



### Note

This filter requires that the input signal be normalized to one.

## Examples

Here is an example of the lpf18 opcode. It uses the file *lpf18.csd* [examples/lpf18.csd].

### Exemple 290. Example of the lpf18 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o lpf18.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a sine waveform.
; Note that its amplitude (kamp) ranges from 0 to 1.
kamp init 1
kcps init 440
knh init 3
ifn = 1
asine buzz kamp, kcps, knh, ifn

; Filter the sine waveform.
; Vary the cutoff frequency (kfco) from 300 to 3,000 Hz.
kfco line 300, p3, 3000
kres init 0.8
kdist init 0.3
aout lpf18 asine, kfco, kres, kdist

out aout * 30000
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for four seconds.
i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Josep M Comajuncosas  
 Spain  
 December 2000

Example written by Kevin Conder with help from Iain Duncan. Thanks goes to Iain for helping with the example.

New in Csound version 4.10

# lpfreson

**lpfreson** — Resynthesises a signal from the data passed internally by a previous **lpread**, applying formant shifting.

## Description

Resynthesises a signal from the data passed internally by a previous **lpread**, applying formant shifting.

## Syntax

```
ares lpfreson asig, kfrqratio
```

## Performance

*asig* -- an audio driving function for resynthesis.

*kfrqratio* -- frequency ratio. Must be greater than 0.

*lpfreson* receives values internally produced by a leading *lpread*. *lpread* gets its values from the control file according to the input value *ktimpnt* (in seconds). If *ktimpnt* proceeds at the analysis rate, time-normal synthesis will result; proceeding at a faster, slower, or variable rate will result in time-warped synthesis. At each *k*-period, *lpread* interpolates between adjacent frames to more accurately determine the parameter values (presented as output) and the filter coefficient settings (passed internally to a subsequent *lpreson*).

The error signal *kerr* (between 0 and 1) derived during predictive analysis reflects the deterministic/random nature of the analyzed source. This will emerge low for pitched (periodic) material and higher for noisy material. The transition from voiced to unvoiced speech, for example, produces an error signal value of about .3. During synthesis, the error signal value can be used to determine the nature of the *lpreson* driving function: for example, by arbitrating between pitched and non-pitched input, or even by determining a mix of the two. In normal speech resynthesis, the pitched input to *lpreson* is a wideband periodic signal or pulse train derived from a unit such as *buzz*, and the nonpitched source is usually derived from *rand*. However, any audio signal can be used as the driving function, the only assumption of the analysis being that it has a flat response.

*lpfreson* is a formant shifted *lpreson*, in which *kfrqratio* is the (cps) ratio of shifted to original formant positions. This permits synthesis in which the source object changes its apparent acoustic size. *lpfreson* with *kfrqratio* = 1 is equivalent to *lpreson*.

Generally, *lpreson* provides a means whereby the time-varying content and spectral shaping of a composite audio signal can be controlled by the dynamic spectral content of another. There can be any number of *lpread*/*lpreson* (or *lpfreson*) pairs in an instrument or in an orchestra; they can read from the same or different control files independently.

## See Also

*lpread*, *lpreson*

# lphasor

lphasor — Génère un indice de table pour la lecture d'échantillons.

## Description

Cet opcode peut être utilisé pour générer un indice de table pour la lecture d'échantillons (par exemple avec `tablexkt`).

## Syntaxe

```
ares lphasor xtrns [, ilps] [, ilpe] [, imode] [, istrt] [, istor]
```

## Initialisation

*ilps* -- début de la boucle.

*ilpe* -- fin de la boucle (doit être supérieur à *ilps* pour que la boucle soit possible). La valeur par défaut de *ilps* et de *ilpe* est zéro.

*imode* (facultatif : 0 par défaut) -- mode de boucle. Les valeurs permises sont :

- 0 : pas de boucle
- 1 : boucle à l'endroit
- 2 : boucle à l'envers
- 3 : boucle à l'endroit et à l'envers

*istrt* (facultatif : 0 par défaut) -- La valeur de sortie initiale (phase). Elle doit être inférieure à *ilpe* si la boucle est active, mais elle peut être supérieure à *ilps* (c-à-d que l'on peut démarrer la lecture au milieu de la boucle).

*istor* (facultatif : 0 par défaut) -- s'il a une valeur différente de zéro l'initialisation est ignorée.

## Exécution

*ares* -- un indice brut de table en échantillons (même unité pour les points de boucle). Peut être utilisé comme indice de table avec les opcodes de table.

*xtrns* -- facteur de transposition, exprimé comme un rapport de pointeur de lecture. *ares* est incrémenté de cette valeur, et répète les valeurs comprises entre les points de boucle. Par exemple, 1.5 signifie une quinte ascendante, 0.75 signifie une quarte descendante. Il ne peut pas être négatif.

## Exemples

Voici un exemple de l'opcode `lphasor`. Il utilise le fichier `lphasor.csd` [examples/lphasor.csd].

## Exemple 291. Exemple de l'opcode lphasor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lphasor.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; Example by Jonathan Murphy Dec 2006

sr      = 44100
ksmps   = 10
nchnls  = 1

instr 1

ifn      = 1      ; table number
ilen     = nsamp(ifn) ; return actual number of samples in table
itrns    = 1      ; no transposition
ilps     = 0      ; loop starts at index 0
ilpe     = ilen ; ends at value returned by nsamp above
imode    = 3      ; loop forwards & backwards
istrt    = 10000 ; commence playback at index 10000 samples
; lphasor provides index into f1
alphs    lphasor   itrns, ilps, ilpe, imode, istrt
atab     tablei    alphs, ifn
; amplify signal
atab     = atab * 10000

out      atab

endin

</CsInstruments>
<CsScore>
f 1 0 262144 1 "beats.wav" 0 4 1
il 0 60
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Istvan Varga  
Janvier 2002  
Exemple écrit par Jonathan Murphy

Nouveau dans la version 4.18

Mise à jour en avril 2002 et en novembre 2002 par Istvan Varga



# lpinterp

lpslot, lpinterp — Computes a new set of poles from the interpolation between two analysis.

## Description

Computes a new set of poles from the interpolation between two analysis.

## Syntax

```
lpinterp islot1, islot2, kmix
```

## Initialization

*islot1* -- slot where the first analysis was stored

*islot2* -- slot where the second analysis was stored

*kmix* -- mix value between the two analysis. Should be between 0 and 1. 0 means analysis 1 only. 1 means analysis 2 only. Any value in between will produce interpolation between the filters.

*lpinterp* computes a new set of poles from the interpolation between two analysis. The poles will be stored in the current *lpslot* and used by the next *lpreson* opcode.

## Examples

Here is a typical orc using the opcodes:

```
ipower init 50000 ; Define sound generator
ifreq  init 440
asrc  buzz ipower,ifreq,10,1

ktime  line 0,p3,p3      ; Define time lin
      lpslot 0          ; Read square data poles
krmsr,krms0,kerr,kcps lpread  ktime,"square.pol"
      lpslot 1          ; Read triangle data poles
krmsr,krms0,kerr,kcps lpread  ktime,"triangle.pol"
kmix  line 0,p3,1        ; Compute result of mixing
      lpinterp 0,1,kmix ; and balance power
ares  lpreson asrc
aout  balance ares,asrc
      out aout
```

## See Also

*lpslot*

## Credits

Author: Gabriel Maldonado

# lposcil

lposcil, lposcil3 — Lit un son échantillonné depuis une table avec boucle optionnelle et haute précision.

## Description

Lit un son échantillonné (mono ou stéréo) depuis une table, avec boucles de soutien et de relâchement optionnelles, et haute précision.

## Syntaxe

```
ares lposcil kamp, kfregratio, kloop, kend, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction

## Exécution

*kamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 1.5 = une quinte ascendante, 0.5 = une octave descendante)

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcil* (looping precise oscillator) permet de faire varier au taux-k le début et la fin d'un son échantillonné contenu dans une table (*GEN01*). Peut être utile pour lire une boucle d'échantillons depuis une table d'onde, avec une vitesse de répétition variant durant l'exécution.

## Voir Aussi

*lposcil3*, *lposcila*, *lposcilsa*, *lposcilsa2*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.52 de Csound

# lposcil3

lposcil3 — Lit un son échantillonné depuis une table en haute précision avec interpolation cubique.

## Description

Lit un son échantillonné (mono ou stéréo) depuis une table, avec boucles de soutien et de relâchement optionnelles, et haute précision. *lposcil3* utilise l'interpolation cubique.

## Syntaxe

```
ares lposcil3 kamp, kfregratio, kloop, kend, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction

## Performance

*kamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 1.5 = une quinte ascendante, 0.5 = une octave descendante)

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcil3* (looping precise oscillator) permet de faire varier au taux-k le début et la fin d'un son échantillonné contenu dans une table (*GEN01*). Peut être utile pour lire une boucle d'échantillons depuis une table d'onde, avec une vitesse de répétition variant durant l'exécution.

## Voir Aussi

*lposcil*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.52 de Csound

# lposcila

*lposcila* — Lit un son échantillonné depuis une table avec boucle optionnelle et haute précision.

## Description

*lposcila* lit un son échantillonné depuis une table avec boucle optionnelle et haute précision.

## Syntaxe

```
ar lposcila aamp, kfregratio, kloop, kend, ift [,iphs]
```

## Initialisation

*ift* -- numéro de la table de fonction

*iphs* -- phase initiale (en échantillons)

## Exécution

*ar* -- signal de sortie

*aamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 1.5 = une quinte ascendante, 0.5 = une octave descendante)

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcila* est semblable à *lposcil*, mais il a un argument d'amplitude de taux audio (au lieu du taux-k) pour permettre des transitoires d'enveloppe rapides.

## Voir Aussi

*lposcil*, *lposcilsa*, *lposcilsa2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# lposcilsa

*lposcilsa* — Lit un son stéréo échantillonné depuis une table avec boucle optionnelle et haute précision.

## Description

*lposcilsa* lit un son stéréo échantillonné depuis une table avec boucle optionnelle et haute précision.

## Syntaxe

```
ar1, ar2 lposcilsa aamp, kfregratio, kloop, kend, ift [,iphs]
```

## Initialisation

*ift* -- numéro de la table de fonction

*iphs* -- phase initiale (en échantillons)

## Exécution

*ar1, ar2* -- signal de sortie

*aamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 1.5 = une quinte ascendante, 0.5 = une octave descendante)

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcilsa* est semblable à *lposcila*, mais il travaille avec des fichiers stéréo chargés par *GEN01*.

## Voir Aussi

*lposcil, lposcila, lposcilsa2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# lposcilsa2

lposcilsa2 — Lit un son stéréo échantillonné depuis une table avec boucle optionnelle et haute précision.

## Description

*lposcilsa2* lit un son stéréo échantillonné depuis une table avec boucle optionnelle et haute précision.

## Syntaxe

```
ar1, ar2 lposcilsa2 aamp, kfregratio, kloop, kend, ift [,iphs]
```

## Initialisation

*ift* -- numéro de la table de fonction

*iphs* -- phase initiale (en échantillons)

## Exécution

*ar1, ar2* -- signal de sortie

*aamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 2 = une octave ascendante). Seule les nombres entiers sont permis.

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcilsa2* est semblable à *lposcilsa*, mais sans interpolation et il ne travaille qu'avec des valeurs entières de *kfregratio*. Beaucoup plus rapide que *lposcilsa*, il est prévu pour fonctionner principalement avec *kfregratio* = 1, étant dans ce cas un substitut rapide de *soundin*, car le fichier son doit être chargé entièrement en mémoire.

## Voir Aussi

*lposcil*, *lposcila*, *lposcilsa*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# lpread

lpread — Reads a control file of time-ordered information frames.

## Description

Reads a control file of time-ordered information frames.

## Syntax

```
krmsr, krmso, kerr, kcps lpread ktmpnt, ifilcod [, inpoles] [, ifrmrate]
```

## Initialization

*ifilcod* -- integer or character-string denoting a control-file (reflection coefficients and four parameter values) derived from n-pole linear predictive spectral analysis of a source audio signal. An integer denotes the suffix of a file *lp.m*; a character-string (in double quotes) gives a filename, optionally a full pathname. If not fullpath, the file is sought first in the current directory, then in that of the environment variable SADIR (if defined). Memory usage depends on the size of the file, which is held entirely in memory during computation but shared by multiple calls (see also *adsyn*, *pvoc*).

*inpoles* (optional, default=0) -- number of poles in the lpc analysis. It is required only when the control file does not have a header; it is ignored when a header is detected.

*ifrmrate* (optional, default=0) -- frame rate per second in the lpc analysis. It is required only when the control file does not have a header; it is ignored when a header is detected.

## Performance

*lpread* accesses a control file of time-ordered information frames, each containing n-pole filter coefficients derived from linear predictive analysis of a source signal at fixed time intervals (e.g. 1/100 of a second), plus four parameter values:

*krmsr* -- root-mean-square (rms) of the residual of analysis

*krmso* -- rms of the original signal

*kerr* -- the normalized error signal

*kcps* -- pitch in Hz

*ktmpnt* -- The passage of time, in seconds, through the analysis file. *ktmpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

*lpread* gets its values from the control file according to the input value *ktmpnt* (in seconds). If *ktmpnt* proceeds at the analysis rate, time-normal synthesis will result; proceeding at a faster, slower, or variable rate will result in time-warped synthesis. At each k-period, *lpread* interpolates between adjacent frames to more accurately determine the parameter values (presented as output) and the filter coefficient settings (passed internally to a subsequent *lpreson*).

The error signal *kerr* (between 0 and 1) derived during predictive analysis reflects the deterministic/random nature of the analyzed source. This will emerge low for pitched (periodic) material and higher for

noisy material. The transition from voiced to unvoiced speech, for example, produces an error signal value of about .3. During synthesis, the error signal value can be used to determine the nature of the *lpreson* driving function: for example, by arbitrating between pitched and non-pitched input, or even by determining a mix of the two. In normal speech resynthesis, the pitched input to *lpreson* is a wideband periodic signal or pulse train derived from a unit such as *buzz*, and the nonpitched source is usually derived from *rand*. However, any audio signal can be used as the driving function, the only assumption of the analysis being that it has a flat response.

*lpfreson* is a formant shifted *lpreson*, in which *kfrqratio* is the (cps) ratio of shifted to original formant positions. This permits synthesis in which the source object changes its apparent acoustic size. *lpfreson* with *kfrqratio* = 1 is equivalent to *lpreson*.

Generally, *lpreson* provides a means whereby the time-varying content and spectral shaping of a composite audio signal can be controlled by the dynamic spectral content of another. There can be any number of *lpread/lpreson* (or *lpfreson*) pairs in an instrument or in an orchestra; they can read from the same or different control files independently.

## See Also

*lpfreson*, *lpreson*, *LPANAL*



# lpreson

lpreson — Resynthesises a signal from the data passed internally by a previous lpread.

## Description

Resynthesises a signal from the data passed internally by a previous lpread.

## Syntax

```
ares lpreson asig
```

## Performance

*asig* -- an audio driving function for resynthesis.

*lpreson* receives values internally produced by a leading *lpread*. *lpread* gets its values from the control file according to the input value *ktimpnt* (in seconds). If *ktimpnt* proceeds at the analysis rate, time-normal synthesis will result; proceeding at a faster, slower, or variable rate will result in time-warped synthesis. At each k-period, *lpread* interpolates between adjacent frames to more accurately determine the parameter values (presented as output) and the filter coefficient settings (passed internally to a subsequent *lpreson*).

The error signal *kerr* (between 0 and 1) derived during predictive analysis reflects the deterministic/random nature of the analyzed source. This will emerge low for pitched (periodic) material and higher for noisy material. The transition from voiced to unvoiced speech, for example, produces an error signal value of about .3. During synthesis, the error signal value can be used to determine the nature of the *lpreson* driving function: for example, by arbitrating between pitched and non-pitched input, or even by determining a mix of the two. In normal speech resynthesis, the pitched input to *lpreson* is a wideband periodic signal or pulse train derived from a unit such as *buzz*, and the nonpitched source is usually derived from *rand*. However, any audio signal can be used as the driving function, the only assumption of the analysis being that it has a flat response.

*lpfreson* is a formant shifted *lpreson*, in which *kfrqratio* is the (cps) ratio of shifted to original formant positions. This permits synthesis in which the source object changes its apparent acoustic size. *lpfreson* with *kfrqratio* = 1 is equivalent to *lpreson*.

Generally, *lpreson* provides a means whereby the time-varying content and spectral shaping of a composite audio signal can be controlled by the dynamic spectral content of another. There can be any number of *lpread*/*lpreson* (or *lpfreson*) pairs in an instrument or in an orchestra; they can read from the same or different control files independently.

## See Also

*lpfreson*, *lpread*

# lpshold

lpshold — Génère un signal de contrôle constitué de segments tenus.

## Description

Génère un signal de contrôle constitué de segments tenus délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

## Syntaxe

```
ksig lpshold kfreq, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
      [, ktime2] [, kvalue2] [...]
```

## Exécution

*ksig* -- Signal de sortie.

*kfreq* -- Taux de répétition en Hz ou en fraction de Hz.

*ktrig* -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

*ktime0...ktimeN* -- Dates des points ; exprimées en fraction de cycle.

*kvalue0...kvalueN* -- Valeurs des points.

*lpshold* est semblable à *loopseg*, mais il ne peut générer que des segments horizontaux, car il maintient une valeur constante pendant chaque intervalle de temps placé entre *ktimeN* et *ktimeN+1*. Il est utile, entre autres, pour un contrôle mélodique comme celui des vieux séquenceurs analogiques.

## Exemples

Voici un exemple de l'opcode *lpshold*. Il utilise le fichier *lpshold.csd* [exemples/lpshold.csd].

### Exemple 292. Exemple de l'opcode lpshold.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lpshold.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1
instr 1
  kfreq line 1, p3, 20

  klp lpshold kfreq, 0, 0, 0, p3*0.25, 20000, p3*0.75, 0

  a1 oscil klp, 440, 1
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for five seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*loopseg*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

# lpsholdp

lpsholdp — Signaux de contrôle basés sur des segments tenus.

## Description

Génère un signal de contrôle constitué de segments de droite tenus délimités par deux ou plus points spécifiés. L'enveloppe entière peut être parcourue en boucle à une vitesse variable. Chaque coordonnée de segment peut aussi varier au taux-k.

## Syntaxe

```
ksig lpsholdp kphase, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \  
      [, ktime2] [, kvalue2] [...]
```

## Exécution

*ksig* - signal de sortie.

*kphase* -

*kvalue0 ...kvalueN* - valeurs des points.

*ktime0 ...ktimeN* - dates des points exprimées en fractions d'un cycle.

L'opcode *lpsholdp* est semblable à *lpshold* ; la seule différence étant que, à la place de la fréquence, une phase variable est utilisée. Si l'on utilise un phaseur pour obtenir la valeur de la phase, on aura un comportement identique à *lpshold*, mais on peut obtenir des résultats intéressants avec des phases à l'évolution non linéaire, ce qui rend *lpsholdp* plus puissant et plus général que *lpshold*.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5. (Auparavant, disponible seulement dans CsoundAV)

# lpslot

*lpslot* — Selects the slot to be use by further lp opcodes.

## Description

Selects the slot to be use by further lp opcodes.

## Syntax

```
lpslot islot
```

## Initialization

*islot* -- number of slot to be selected.

## Performance

*lpslot* selects the slot to be use by further lp opcodes. This is the way to load and reference several analyses at the same time.

## Examples

Here is a typical orc using the opcodes:

```
ipower init 50000 ; Define sound generator
ifreq init 440
asrc buzz ipower,ifreq,10,1

ktime line 0,p3,p3 ; Define time lin
      lpslot 0 ; Read square data poles
krmsr,krms0,kerr,kcps lpread ktime,"square.pol"
      lpslot 1 ; Read triangle data poles
krmsr,krms0,kerr,kcps lpread ktime,"triangle.pol"
kmix line 0,p3,1 ; Compute result of mixing
      lpinterp 0,1,kmix ; and balance power
ares lpreson asrc
aout balance ares,asrc
      out aout
```

## See Also

*lpinterp*

## Credits

Author: Mark Resibois  
Brussels  
1996

New in version 3.44

## mac

mac — Multiplies and accumulates a- and k-rate signals.

## Description

Multiplies and accumulates a- and k-rate signals.

## Syntax

```
ares mac asig1, ksig1 [, asig2] [, ksig2] [, asig3] [, ksig3] [...]
```

## Performance

*ksig1, etc.* -- k-rate input signals

*asig1, etc.* -- a-rate input signals

*mac* multiplies and accumulates a- and k-rate signals. It is equivalent to:

$$\text{ares} = \text{asig1} * \text{ksig1} + \text{asig2} * \text{ksig2} + \text{asig3} * \text{ksig3} + \dots$$

## See Also

*maca*

## Credits

Author: John ffitch  
University of Bath, Codemist, Ltd.  
Bath, UK  
May 1999

New in Csound version 3.54

# maca

maca — Multiply and accumulate a-rate signals only.

## Description

Multiply and accumulate a-rate signals only.

## Syntax

```
ares maca asig1 , asig2 [, asig3] [, asig4] [, asig5] [...]
```

## Performance

*asig1*, *asig2*, ... -- a-rate input signals

*maca* multiplies and accumulates a-rate signals only. It is equivalent to:

$$\text{ares} = \text{asig1} * \text{asig2} + \text{asig3} * \text{asig4} + \text{asig5} * \text{asig6} + \dots$$

## See Also

*mac*

## Credits

Author: John ffitch  
University of Bath, Codemist, Ltd.  
Bath, UK  
May 1999

New in Csound version 3.54



# madsr

madsr — Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *linsegr*.

## Description

Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *linsegr*.

## Syntaxe

```
ares madsr iatt, idec, islev, irel [, idel] [, ireltim]
```

```
kres madsr iatt, idec, islev, irel [, idel] [, ireltim]
```

## Initialisation

*iatt* -- durée de l'attaque (attack)

*idec* -- durée de la première chute (decay)

*islev* -- niveau d'entretien (sustain)

*irel* -- durée de la chute (release)

*idel* -- délai de niveau zéro avant le démarrage de l'enveloppe

*ireltim* (facultatif, par défaut = -1) -- Contrôle la durée du relâchement après la réception d'un événement MIDI note-off. S'il est inférieur à zéro, la durée de relâchement la plus longue de l'instrument courant est utilisée. S'il est nul ou positif, la valeur donnée sera utilisée comme durée de relâchement. Sa valeur par défaut est -1. (Nouveau dans Csound 3.59 - pas encore entièrement testé).

Noter que la durée du relâchement ne peut pas dépasser 32767/*kr* secondes.

## Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

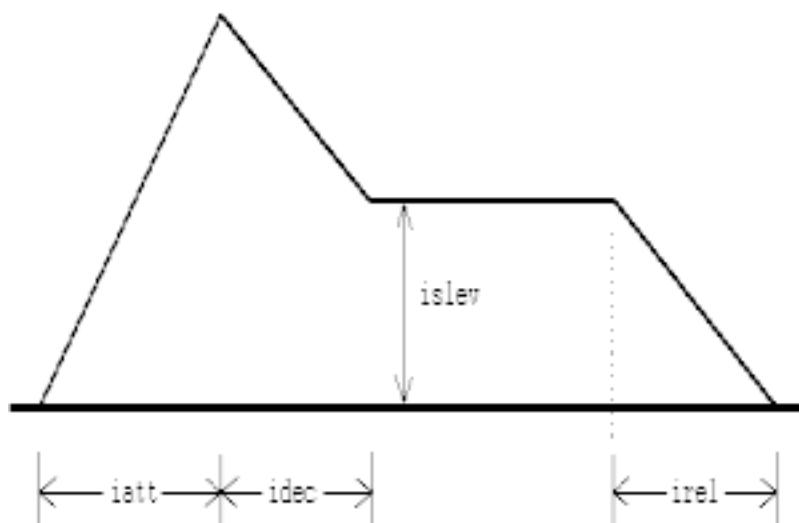


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des événements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et peut donc être utilisé dans les applications MIDI.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note-off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *madsr*, car la durée est allongée automatiquement.



## Note

Les durées pour *iatt*, *idec* et *irel* ne peuvent pas être 0. Si l'on utilise 0, aucune enveloppe n'est générée. Utilisez une très petite valeur comme 0.0001 si vous désirez une attaque, une chute ou un relâchement instantanés.

## Exemples

Voici un exemple de l'opcode *madsr*. Il utilise le fichier *madsr.csd* [examples/madsr.csd].

### Exemple 293. Exemple de l'opcode *madsr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o madsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Iain McCurdy */
; Initialize the global variables.
sr = 44100
kr = 441
```

```
ksmps = 100
nchnls = 1

; Instrument #1.
instr 1
; Attack time.
iattack = 0.5
; Decay time.
idecay = 0
; Sustain level.
isustain = 1
; Release time.
irelease = 0.5
aenv madsr iattack, idecay, isustain, irelease

a1 oscili 10000, 440, 1
out a1*aenv
endin

</CsInstruments>
<CsScore>

/* Written by Iain McCurdy */
; Table #1, a sine wave.
f 1 0 1024 10 1

; Leave the score running for 6 seconds.
f 0 6

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*linsegr, expsegr, envlpxr, mxadsr, madsr, xadsr expon, expseg, expsega line, linseg, xtratim*

## Crédits

Auteur : John ffitch

Novembre 2002. Merci à Rasmus Ekman pour avoir documenté le paramètre *ireltim*.

Décembre 2002. Merci à Iain McCurdy pour avoir ajouté un exemple.

Décembre 2002. Merci à Istvan Varga pour avoir indiqué la durée maximale de relâchement.

Nouveau dans la version 3.49 de Csound.

# mandel

mandel — Ensemble de Mandelbrot.

## Description

Retourne le nombre d'itérations correspondant à un point donné du plan complexe auquel on applique les formules de l'ensemble de Mandelbrot.

## Syntaxe

```
kiter, koutrig mandel ktrig, kx, ky, kmaxIter
```

## Exécution

*kiter* - nombre d'itérations

*koutrig* - signal de déclenchement en sortie

*ktrig* - signal de déclenchement en entrée

*kx, ky* - coordonnées d'un point appartenant au plan complexe

*kmaxIter* - nombre maximum d'itérations autorisé

*mandel* est un opcode qui utilise les formules de l'ensemble de Mandelbrot pour générer une sortie que l'on peut appliquer à n'importe quel paramètre musical (ou non musical). Il comprend deux paramètres de sortie : *kiter*, qui contient le nombre d'itérations d'un point donné, et *koutrig*, qui génère un 'bang' de déclenchement chaque fois que *kiter* change. L'évaluation d'un nouveau nombre d'itérations n'a lieu que lorsque *ktrig* prend une valeur non nulle. Les coordonnées du plan complexe sont fixées dans *kx* et *ky*, tandis que *kmaxIter* contient le nombre maximum d'itérations. Les valeurs de sorties, qui sont des nombres entiers, peuvent être interprétées de n'importe quelle manière par le compositeur.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Seulement disponible auparavant dans CsoundAV)

# mandol

mandol — Une simulation de mandoline.

## Description

Une simulation de mandoline.

## Syntaxe

```
ares mandol kamp, kfreq, kpluck, kdetune, kgain, ksize, ifn [, iminfreq]
```

## Initialisation

*ifn* -- numéro d'une table contenant la forme d'onde du pincement de corde. Le fichier *mandpluk.aiff* [examples/mandpluk.aiff] convient pour cela. On peut aussi l'obtenir à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

*iminfreq* (facultatif, 0 par défaut) -- Fréquence la plus basse pour une note. Si ce paramètre est omis, il prend la valeur initiale de *kfreq*.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kpluck* -- Position du pincement sur la corde, compris entre 0 et 1. Valeur suggérée : 0,4.

*kdetune* -- Proportion de désaccord entre les deux cordes. La valeur suggérée va de 0,9 à 1.

*kgain* -- le gain de la boucle du modèle, compris entre 0,97 et 1.

*ksize* -- La taille du corps de la mandoline. Compris entre 0 et 2.

## Exemples

Voici un exemple de l'opcode mandol. Il utilise les fichiers *mandol.csd* [examples/mandol.csd], et *mandpluk.aiff* [examples/mandpluk.aiff].

### Exemple 294. Exemple de l'opcode mandol.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o mandol.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; kamp = 30000
; kfreq = 880
; kpluck = 0.4
; kdetune = 0.99
; kgain = 0.99
; ksize = 2
; ifn = 1
; ifreq = 220

a1 mandol 30000, 880, 0.4, 0.99, 0.99, 2, 1, 220

out a1
endin

</CsInstruments>
<CsScore>

; Table #1: the "mandpluk.aiff" audio file
f 1 0 8192 1 "mandpluk.aiff" 0 0 0

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# marimba

marimba — Modèle physique de la frappe d'un bloc de bois.

## Description

La sortie audio est un son tel que celui produit lorsque l'on frappe un bloc de bois comme dans un marimba. Il s'agit d'un modèle physique développé d'après Perry Cook mais recodé pour Csound.

## Syntaxe

```
ares marimba kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec \
      [, idoubles] [, itriples]
```

## Initialisation

*ihrd* -- la dureté de la baguette utilisée pour la frappe. On utilise un intervalle allant de 0 à 1. 0,5 est une valeur adéquate.

*ipos* -- le point d'impact sur le bloc, compris entre 0 et 1.

*imp* -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

*ivfn* -- forme du vibrato, habituellement une table sinus, créée par une fonction

*idec* -- durée avant la fin de la note lorsqu'il y a une atténuation

*idoubles* (facultatif) -- pourcentage de frappes doubles. La valeur par défaut est de 40%.

*itriples* (facultatif) -- pourcentage de frappes triples. La valeur par défaut est de 20%.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kvibf* -- Fréquence du vibrato en Hertz. L'intervalle conseillé va de 0 à 12.

*kvamp* -- Amplitude du vibrato.

## Exemples

Voici un exemple de l'opcode marimba. Il utilise les fichiers *marimba.csd* [examples/marimba.csd] et *marmstk1.wav* [examples/marmstk1.wav].

### Exemple 295. Exemple de l'opcode marimba.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o marimba.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 128
nchnls = 2

; Instrument #1.
instr 1
  ifreq = cpspch(p4)
  ihrd = 0.1
  ipos = 0.561
  imp = 1
  kvibf = 6.0
  kvamp = 0.05
  ivibfn = 2
  idec = 0.6

  a1 marimba 20000, ifreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec, 20, 10

  outs a1, a1
endin

</CsInstruments>
<CsScore>

; Table #1, the "marmstkl.wav" audio file.
f 1 0 256 1 "marmstkl.wav" 0 0 0
; Table #2, a sine wave for the vibrato.
f 2 0 128 10 1

; Play Instrument #1 for one second.
i 1 0 1 8.09
i 1 + 0.5 8.00
i 1 + 0.5 7.00
i 1 + 0.25 8.02
i 1 + 0.25 8.01
i 1 + 0.25 7.09
i 1 + 0.25 8.02
i 1 + 0.25 8.01
i 1 + 0.25 7.09
i 1 + 0.3333 8.09
i 1 + 0.3333 8.02
i 1 + 0.3334 8.01
i 1 + 0.25 8.00
i 1 + 0.3333 8.09
i 1 + 0.3333 8.02
i 1 + 0.25 8.01
i 1 + 0.3333 7.00
i 1 + 0.3334 6.00

e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*vibes*

## Crédits



Auteur : John ffitch (d'après Perry Cook)  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# massign

massign — Affecte un numéro de canal MIDI à un instrument de Csound.

## Description

Affecte un numéro de canal MIDI à un instrument de Csound.

## Syntaxe

```
massign ichnl, insnum[, ireset]
```

```
massign ichnl, "insname"[, ireset]
```

## Initialisation

*ichnl* -- numéro de canal MIDI (1-16).

*insnum* -- numéro de l'instrument d'orchestre de Csound. S'il est inférieur ou égal à zéro, le canal est désactivé (c-à-d. qu'il ne déclenche aucun instrument de csound, bien que l'information soit toujours reçue par des opcodes tels que *midin*).

« *insname* » -- une chaîne de caractères entre guillemets représentant un nom d'instrument.

*ireset* -- si est non nul, les contrôleurs sont réinitialisés ; c'est le comportement par défaut.

## Exécution

Affecte un numéro de canal MIDI à un instrument de Csound. Egalement utile pour s'assurer qu'un instrument particulier (si son numéro est compris entre 1 et 16) ne sera pas déclenché par des messages MIDI noteon (si l'on utilise quelque chose comme *midin* pour interpréter l'information MIDI). Dans ce cas, fixer *insnum* à un nombre inférieur ou égal à 0.

Si *ichan* est fixé à 0, la valeur de *insnum* est utilisée pour tous les canaux. On peut envoyer de cette manière tous les canaux MIDI vers un seul instrument de Csound. On peut aussi empêcher le déclenchement des instruments à partir d'événements de note MIDI en provenance de tous les canaux avec la ligne suivante :

```
massign 0, 0
```

Ceci peut être utile si l'on effectue toutes les évaluations MIDI dans Csound avec un instrument actif en permanence (par exemple en utilisant *midin* et *turnon*) pour éviter une doublure de l'instrument quand une note est jouée.

## Voir Aussi

*ctrlinit*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT, Cambridge, Mass.

Nouveau dans la version 3.47 de Csound

Le paramètre *ireset* est nouveau dans Csound5

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de canal MIDI.

# max

max — Produces a signal that is the maximum of any number of input signals.

## Description

The *max* opcode takes any number of a-rate or k-rate signals as input (all of the same rate), and outputs a signal at the same rate that is the maximum of all of the inputs. For a-rate signals, the inputs are compared one sample at a time (i.e. *max* does not scan an entire ksmps period of a signal for its local maximum as the *max\_k* opcode does).

## Syntax

```
amax max ain1 [, ain2] [, ain3] [, ain4] [...]
```

```
kmax max kin1 [, kin2] [, kin3] [, kin4] [...]
```

## Performance

*ain1, ain2, ...* -- a-rate signals to be compared.

*kin1, kin2, ...* -- k-rate signals to be compared.

## See Also

*min, maxabs, minabs, maxaccum, minaccum, maxabsaccum, minabsaccum, max\_k*

## Credits

Author: Anthony Kozar  
March 2006

New in Csound version 5.01

# maxabs

maxabs — Produces a signal that is the maximum of the absolute values of any number of input signals.

## Description

The *maxabs* opcode takes any number of a-rate or k-rate signals as input (all of the same rate), and outputs a signal at the same rate that is the maximum of all of the inputs. It is identical to the *max* opcode except that it takes the absolute value of each input before comparing them. Therefore, the output is always non-negative. For a-rate signals, the inputs are compared one sample at a time (i.e. *maxabs* does not scan an entire ksmpls period of a signal for its local maximum as the *max\_k* opcode does).

## Syntax

```
amax maxabs ain1 [, ain2] [, ain3] [, ain4] [...]
```

```
kmax maxabs kin1 [, kin2] [, kin3] [, kin4] [...]
```

## Performance

*ain1, ain2, ...* -- a-rate signals to be compared.

*kin1, kin2, ...* -- k-rate signals to be compared.

## See Also

*minabs, max, min, maxaccum, minaccum, maxabsaccum, minabsaccum, max\_k*

## Credits

Author: Anthony Kozar  
March 2006

New in Csound version 5.01

# maxabsaccum

maxabsaccum — Accumulates the maximum of the absolute values of audio signals.

## Description

*maxabsaccum* compares two audio-rate variables and stores the maximum of their absolute values into the first.

## Syntax

```
maxabsaccum aAccumulator, aInput
```

## Performance

*aAccumulator* -- audio variable to store the maximum value

*aInput* -- signal that *aAccumulator* is compared to

The *maxabsaccum* opcode is designed to accumulate the maximum value from among many audio signals that may be in different note instances, different channels, or otherwise cannot all be compared at once using the *maxabs* opcode. *maxabsaccum* is identical to *maxaccum* except that it takes the absolute value of *aInput* before the comparison. Its semantics are similar to *vincr* since *aAccumulator* is used as both an input and an output variable, except that *maxabsaccum* keeps the maximum absolute value instead of adding the signals together. *maxabsaccum* performs the following operation on each pair of samples:

$$\text{if } (\text{abs}(\text{aInput}) > \text{aAccumulator}) \text{ aAccumulator} = \text{abs}(\text{aInput})$$

*aAccumulator* will usually be a global audio variable. At the end of any given computation cycle (k-period), after its value is read and used in some way, the accumulator variable should usually be reset to zero (perhaps by using the *clear* opcode). Clearing to zero is sufficient for *maxabsaccum*, unlike the *maxaccum* opcode.

## See Also

*minabsaccum*, *maxaccum*, *minaccum*, *max*, *min*, *maxabs*, *minabs*, *vincr*, *clear*

## Credits

Author: Anthony Kozar  
March 2006

New in Csound version 5.01

# maxaccum

maxaccum — Accumulates the maximum value of audio signals.

## Description

*maxaccum* compares two audio-rate variables and stores the maximum value between them into the first.

## Syntax

```
maxaccum aAccumulator, aInput
```

## Performance

*aAccumulator* -- audio variable to store the maximum value

*aInput* -- signal that aAccumulator is compared to

The *maxaccum* opcode is designed to accumulate the maximum value from among many audio signals that may be in different note instances, different channels, or otherwise cannot all be compared at once using the *max* opcode. Its semantics are similar to *vincr* since *aAccumulator* is used as both an input and an output variable, except that *maxaccum* keeps the maximum value instead of adding the signals together. *maxaccum* performs the following operation on each pair of samples:

```
if (aInput > aAccumulator) aAccumulator = aInput
```

*aAccumulator* will usually be a global audio variable. At the end of any given computation cycle (k-period), after its value is read and used in some way, the accumulator variable should usually be reset to zero (perhaps by using the *clear* opcode). Care must be taken however if *aInput* is negative at any point, in which case the accumulator should be initialized and reset to some large enough negative value that will always be less than the input signals to which it is compared.

## See Also

*minaccum, maxabsaccum, minabsaccum, max, min, maxabs, minabs, vincr, clear*

## Credits

Author: Anthony Kozar  
March 2006

New in Csound version 5.01

# maxalloc

maxalloc — Limite le nombre d'allocations pour un instrument.

## Description

Limite le nombre d'allocations pour un instrument.

## Syntaxe

```
maxalloc insnum, icount
```

## Initialisation

*insnum* -- numéro de l'instrument

*icount* -- nombre d'allocations de l'instrument

## Exécution

Toutes les instances de *maxalloc* doivent être définies dans la section d'en-tête, pas dans le corps de l'instrument.

## Exemples

Voici un exemple de l'opcode maxalloc. Il utilise le fichier *maxalloc.csd* [examples/maxalloc.csd].

### Exemple 296. Exemple de l'opcode maxalloc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o maxalloc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Limit Instrument #1 to three instances.
maxalloc 1, 3

; Instrument #1
instr 1
; Generate a waveform, get the cycles per second from the 4th p-field.
a1 oscil 6500, p4, 1
out a1
endin
```



```
</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play five instances of Instrument #1 for one second.
; Note that 4th p-field contains cycles per second.
i 1 0 1 220
i 1 0 1 440
i 1 0 1 880
i 1 0 1 1320
i 1 0 1 1760
e

</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra un message comme celui-ci :

```
WARNING: cannot allocate last note because it exceeds instr maxalloc
```

## Voir Aussi

*cpuprc, prealloc*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# max\_k

max\_k — Local maximum (or minimum) value of an incoming *asig* signal

## Description

*max\_k* outputs the local maximum (or minimum) value of the incoming *asig* signal, checked in the time interval between *ktrig* has become true twice.

## Syntax

```
knumkout max_k asig, ktrig, itype
```

## Initialization

*itype* - itype determinates the behaviour of max\_k (see below)

## Performance

*asig* - incoming (input) signal

*ktrig* - trigger signal

max\_k outputs the local maximum (or minimum) value of the incoming *asig* signal, checked in the time interval between *ktrig* has become true twice. *itype* determinates the behaviour of max\_k:

1 - absolute maximum (sign of negative values is changed to positive before evaluation)

2 - actual maximum

3 - actual minimum

4 - calculate average value of *asig* in the time interval

This opcode can be useful in several situations, for example to implement a vu-meter.

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# mclock

mclock — Sends a MIDI CLOCK message.

## Description

Sends a MIDI CLOCK message.

## Syntax

```
mclock ifreq
```

## Initialization

*ifreq* -- clock message frequency rate in Hz

## Performance

Sends a MIDI CLOCK message (0xF8) every  $1/ifreq$  seconds. So *ifreq* is the frequency rate of CLOCK message in Hz.

## See Also

*mrtmsg*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

# mdelay

mdelay — Un opcode de délai MIDI.

## Description

Un opcode de délai MIDI.

## Syntaxe

**mdelay** *kstatus*, *kchan*, *kd1*, *kd2*, *kdelay*

## Exécution

*kstatus* -- octet d'état du message MIDI message à retarder.

*kchan* -- canal MIDI (1-16)

*kd1* -- premier octet de donnée MIDI

*kd2* -- deuxième octet de donnée MIDI

*kdelay* -- délai en secondes

Chaque fois que *kstatus* est différent zéro, *mdelay* envoie un message MIDI sur le port de sortie MIDI après *kdelay* secondes. Cet opcode est utile pour implémenter des délais MIDI. Il peut y avoir plusieurs instances de *mdelay* dans le même instrument avec des valeurs d'argument différentes, si bien que l'on peut implémenter des echos MIDI complexes et colorés. De plus, on peut changer la durée du retard au taux-k.

## Exemples

Voici un exemple de l'opcode mdelay. Il utilise le fichier *mdelay.csd* [examples/mdelay.csd].

### Exemple 297. Exemple de l'opcode mdelay.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d          -M0  -Q1;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ;Triggered by MIDI notes on channel 1
```

```
kstatus init 0
ifund notnum
ivel veloc

noteondur 1, ifund, ivel, 1

kstatus = kstatus + 1

idel1 = .2
idel2 = .4
idel3 = .6
idel4 = .8

;make four delay lines

mdelay      kstatus,1,ifund+2, ivel,idel1
mdelay      kstatus,1,ifund+4, ivel,idel2
mdelay      kstatus,1,ifund+6, ivel,idel3
mdelay      kstatus,1,ifund+8, ivel,idel4

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Novembre 1998

Nouveau dans la version 3.492 de Csound

# metro

metro — Trigger Metronome

## Description

Generate a metronomic signal to be used in any circumstance an isochronous trigger is needed.

## Syntax

```
ktrig metro kfreq [, initphase]
```

## Initialization

*initphase* - initial phase value (in the 0 to 1 range)

## Performance

*ktrig* - output trigger signal

*kfreq* - frequency of trigger bangs in cps

*metro* is a simple opcode that outputs a sequence of isochronous bangs (that is 1 values) each 1/kfreq seconds. Trigger signals can be used in any circumstance, mainly to temporize realtime algorithmic compositional structures.



### Note

*metro* will produce a trigger signal of 1 when its phase is exactly 0 or 1. If you want to skip the initial trigger, use a very small value like 0.00000001.

## Examples

Here is an example of the metro opcode. It uses the file *metro.csd* [examples/metro.csd]

### Exemple 298. Example of the metro opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      100
ksmps   =      441
nchnls  =      2

      instr    1
ktrig metro 0.2
printk2 ktrig
      endin
```

```
</CsInstruments>  
<CsScore>  
i 1 0 20  
  
</CsScore>  
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado.

Example written by Andrés Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# midic14

**midic14** — Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest midic14 ictlno1, ictlno2, imin, imax [, ifn]
```

```
kdest midic14 ictlno1, ictlno2, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*imin* -- valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- valeur décimale maximale de sortie définie par l'utilisateur

*midic14* (contrôle MIDI sur 14 bit au taux-i et au taux-k) permet un signal MIDI sur 14 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table interpolée. Il nécessite deux contrôleurs MIDI en entrée.



### Note

Veuillez noter que la famille des opcodes *midic* est prévue pour des événements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un événement de partition.



## Voir aussi

*ctrl7, ctrl14, ctrl21, initc7, initc14, initc21, midic7, midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

# midic21

**midic21** — Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest midic21 ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
```

```
kdest midic21 ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids moyen (0-127)

*ictlno3* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*imin* -- valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- valeur décimale maximale de sortie définie par l'utilisateur

*midic21* (contrôle MIDI sur 21 bit au taux-i et au taux-k) permet un signal MIDI sur 21 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table interpolée. Il nécessite trois contrôleurs MIDI en entrée.



### Note

Veuillez noter que la famille des opcodes *midic* est prévue pour des événements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un événement de partition.

## Voir aussi

*ctrl7, ctrl14, ctrl21, initc7, initc14, initc21, midic7, midic14*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

# midic7

*midic7* — Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest midic7 ictlno, imin, imax [, ifn]
```

```
kdest midic7 ictlno, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ictlno* -- numéro de contrôleur MIDI (0-127)

*imin* -- valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imin* et *imax*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- valeur décimale maximale de sortie définie par l'utilisateur

*midic7* (contrôle MIDI sur 7 bit au taux-i et au taux-k) permet un signal MIDI sur 7 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Il permet également en option une indexation de table sans interpolation. Dans *midic7* les valeurs minimale et maximale peuvent varier au taux-k.



### Note

Veillez noter que la famille des opcodes *midic* est prévue pour des événements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un événement de partition.

## Voir aussi

*ctrl7*, *ctrl14*, *ctrl21*, *initc7*, *initc14*, *initc21*, *midic14*, *midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

# midichannelaftertouch

midichannelaftertouch — Gets a MIDI channel's aftertouch value.

## Description

*midichannelaftertouch* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midichannelaftertouch xchannelaftertouch [, ilow] [, ihigh]
```

## Initialization

*ilow* (optional) -- optional low value after rescaling, defaults to 0.

*ihigh* (optional) -- optional high value after rescaling, defaults to 127.

## Performance

*xchannelaftertouch* -- returns the MIDI channel aftertouch during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the value of *xchannelaftertouch* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the value of *xchannelaftertouch* remains unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## Examples

Here is an example of the *midichannelaftertouch* opcode. It uses the file *midichannelaftertouch.csd* [examples/midichannelaftertouch.csd].

## Exemple 299. Example of the midichannelaftertouch opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midichannelaftertouch.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kaft init 0
  midichannelaftertouch kaft

  ; Display the aftertouch value when it changes.
  printk2 kaft
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
i1 127.00000
i1 20.00000
i1 44.00000
```

## See Also

*midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

## Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

# midichn

midichn — Returns the MIDI channel number from which the note was activated.

## Description

*midichn* returns the MIDI channel number (1 - 16) from which the note was activated. In the case of score notes, it returns 0.

## Syntax

```
ichn midichn
```

## Initialization

*ichn* -- channel number. If the current note was activated from score, it is set to zero.

## Examples

Here is a simple example of the midichn opcode. It uses the file *midichn.csd* [examples/midichn.csd].

### Exemple 300. Example of the midichn opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midichn.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  il midichn

  print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e

</CsScore>
</CsoundSynthesizer>
```



Here is an advanced example of the `midichn` opcode. It uses the file `midichn_advanced.csd` [examples/midichn\_advanced.csd].

Don't forget that you must include the `-F` flag when using an external MIDI file like « `midichn_advanced.mid` ».

### Exemple 301. An advanced example of the `midichn` opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midichn_advanced.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 1

        massign 1, 1          ; all channels use instr 1
        massign 2, 1
        massign 3, 1
        massign 4, 1
        massign 5, 1
        massign 6, 1
        massign 7, 1
        massign 8, 1
        massign 9, 1
        massign 10, 1
        massign 11, 1
        massign 12, 1
        massign 13, 1
        massign 14, 1
        massign 15, 1
        massign 16, 1

gicnt = 0          ; note counter

        instr 1

gicnt = gicnt + 1 ; update note counter
kcnt init gicnt ; copy to local variable
ichn midichn          ; get channel number
istime times          ; note-on time

        if (ichn > 0.5) goto 12          ; MIDI note
printks "note %.0f (time = %.2f) was activated from the score\\n", \
        3600, kcnt, istime
        goto 11

12:
        printks "note %.0f (time = %.2f) was activated from channel %.0f\\n", \
        3600, kcnt, istime, ichn

11:
        endin

</CsInstruments>
<CsScore>

t 0 60
f 0 6 2 -2 0
i 1 1 0.5
i 1 4 0.5
e

</CsScore>
```

```
</CsoundSynthesizer>
```

Its output should include lines like:

```
note 7 (time = 0.00) was activated from channel 4  
note 8 (time = 0.00) was activated from channel 2
```

## See Also

*pgmassign*

## Credits

Author: Istvan Varga  
May 2002

The simple example was written by Kevin Conder.

New in version 4.20

# midicontrolchange

midicontrolchange — Gets a MIDI control change value.

## Description

*midicontrolchange* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midicontrolchange xcontroller, xcontrollervalue [, ilow] [, ihigh]
```

## Initialization

*ilow* (optional) -- optional low value after rescaling, defaults to 0.

*ihigh* (optional) -- optional high value after rescaling, defaults to 127.

## Performance

*xcontroller* -- specifies a MIDI controller number (0-127).

*xcontrollervalue* -- returns the value of the MIDI controller during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of the *xcontroller* and *xcontrollervalue* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xcontroller* and *xcontrollervalue* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## See Also

*midichannelaftertouch*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

## Credits

Author: Michael Gogins

New in version 4.20

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# midictrl

midictrl — Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

## Description

Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

## Syntaxe

```
ival midictrl inum [, imin] [, imax]
```

```
kval midictrl inum [, imin] [, imax]
```

## Initialisation

*inum* -- numéro de contrôleur MIDI (0-127)

*imin, imax* -- Ajuste les limites minimale et maximales pour les valeurs obtenues.

## Exécution

Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

## Avertissement

*midictrl* doit être utilisé seulement pour les notes déclenchées par MIDI, permettant la disponibilité d'un numéro de canal associé. Pour les notes activées depuis la partition, les événements de ligne, ou l'orchestre, il faut utiliser l'opcode *ctrl7* qui prend un numéro de canal explicite.

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

# mididefault

mididefault — Changes values, depending on MIDI activation.

## Description

*mididefault* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
mididefault xdefault, xvalue
```

## Performance

*xdefault* -- specifies a default value that will be used during MIDI activation.

*xvalue* -- overwritten by *xdefault* during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode will overwrite the value of *xvalue* with the value of *xdefault*. If the instrument was *NOT* activated by MIDI input, *xvalue* will remain unchanged.

This enables score pfields to receive a default value during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## See Also

*midichannelaftertouch*, *midicontrolchange*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*, *midiprogramchange*

## Credits

Author: Michael Gogins

New in version 4.20

# midin

midin — Returns a generic MIDI message received by the MIDI IN port.

## Description

Returns a generic MIDI message received by the MIDI IN port

## Syntax

```
kstatus, kchan, kdata1, kdata2 midin
```

## Performance

*kstatus* -- the type of MIDI message. Can be:

- 128 (note off)
- 144 (note on)
- 160 (polyphonic aftertouch)
- 176 (control change)
- 192 (program change)
- 208 (channel aftertouch)
- 224 (pitch bend)
- 0 if no MIDI message are pending in the MIDI IN buffer

*kchan* -- MIDI channel (1-16)

*kdata1*, *kdata2* -- message-dependent data values

*midin* has no input arguments, because it reads at the MIDI in port implicitly. It works at k-rate. Normally (i.e., when no messages are pending) *kstatus* is zero, only when MIDI data are present in the MIDI IN buffer, is *kstatus* set to the type of the relevant messages.



### Note

Be careful when using *midin* in low numbered instruments, since a MIDI note will launch additional instances of the instrument, resulting in duplicate events and weird behaviour. Use *massign* to direct MIDI note on messages to a different instrument or to disable triggering of instruments from MIDI.

## Examples

Here is an example of the *midin* opcode. It uses the file *midin.csd* [examples/midin.csd].

## Exemple 302. Example of the midiin opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      -M0  -+rtmidi=virtual ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midiin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 10
nchnls  = 1

; Example by schwaahed 2006

      massign      0, 130 ; make sure that all channels
      pgmassign    0, 130 ; and programs are assigned to test instr

      instr      130

      knotlength   init      0
      knoteontime   init      0

      kstatus, kchan, kdata1, kdata2          midiin

      if (kstatus == 128) then
      knoteofftime   times
      knotlength     =      knoteofftime - knoteontime
      printks "kstatus= %d, kchan = %d, \\tnote#   = %d, velocity = %d \\tNote OFF\\t%f %f\\n", 0, kstatus, kchan, kdata1, kdata2

      elseif (kstatus == 144) then
      knoteontime     times
      printks "kstatus= %d, kchan = %d, \\tnote#   = %d, velocity = %d \\tNote ON\\t%f\\n", 0, kstatus, kchan, kdata1, kdata2

      elseif (kstatus == 160) then
      printks "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tPolyphonic Aftertouch\\n", 0, kstatus, kchan, kdata1, kdata2

      elseif (kstatus == 176) then
      printks "kstatus= %d, kchan = %d, \\t CC = %d, value = %d \\tControl Change\\n", 0, kstatus, kchan, kdata1, kdata2

      elseif (kstatus == 192) then
      printks "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tProgram Change\\n", 0, kstatus, kchan, kdata1, kdata2

      elseif (kstatus == 208) then
      printks "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tChannel Aftertouch\\n", 0, kstatus, kchan, kdata1, kdata2

      elseif (kstatus == 224) then
      printks "kstatus= %d, kchan = %d, \\t ( data1 , kdata2 ) = ( %d, %d )\\tPitch Bend\\n", 0, kstatus, kchan, kdata1, kdata2

      endif

      endin

</CsInstruments>
<CsScore>
i130 0 3600
e
</CsScore>
</CsoundSynthesizer>
```

## Credits



Author: Gabriel Maldonado  
Italy  
1998

New in Csound version 3.492

# midinoteoff

midinoteoff — Gets a MIDI noteoff value.

## Description

*midinoteoff* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midinoteoff xkey, xvelocity
```

## Performance

*xkey* -- returns MIDI key during MIDI activation, remains unchanged otherwise.

*xvelocity* -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of the *xkey* and *xvelocity* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xkey* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## Examples

Here is an example of the *midinoteoff* opcode. It uses the file *midinoteoff.csd* [examples/midinoteoff.csd].

### Exemple 303. Example of the midinoteoff opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midinoteoff.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kkey init 0
  kvelocity init 0

  midinoteoff kkey, kvelocity

  ; Display the key value when it changes.
  printk2 kkey
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

Its output should include lines like:

```

i1      60.00000
i1      76.00000

```

## See Also

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoncps, midinoteonkey, midinoteonoct, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

# midinoteoncps

midinoteoncps — Gets a MIDI note number as a cycles-per-second frequency.

## Description

*midinoteoncps* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midinoteoncps xcps, xvelocity
```

## Performance

*xcps* -- returns MIDI key translated to cycles per second during MIDI activation, remains unchanged otherwise.

*xvelocity* -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xcps* and *xvelocity* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xcps* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## Examples

Here is an example of the midinoteoncps opcode. It uses the file *midinoteoncps.csd* [examples/midinoteoncps.csd].

### Exemple 304. Example of the midinoteoncps opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime output leave only the line below:
; -o midinoteoncps.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kcps init 0
  kvelocity init 0

  midinoteoncps kcps, kvelocity

  ; Display the cycles-per-second value when it changes.
  printk2 kcps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

Its output should include lines like:

```

i1 261.62561
i1 440.00006

```

## See Also

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteonkey, midinoteonoct, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

# midinoteonkey

midinoteonkey — Gets a MIDI note number value.

## Description

*midinoteonkey* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midinoteonkey xkey, xvelocity
```

## Performance

*xkey* -- returns MIDI key during MIDI activation, remains unchanged otherwise.

*xvelocity* -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xkey* and *xvelocity* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xkey* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## Examples

Here is an example of the midinoteonkey opcode. It uses the file *midinoteonkey.csd* [examples/midinoteonkey.csd].

### Exemple 305. Example of the midinoteonkey opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime output leave only the line below:
; -o midinoteonkey.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kkey init 0
  kvelocity init 0

  midinoteonkey kkey, kvelocity

  ; Display the key value when it changes.
  printk2 kkey
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

Its output should include lines like:

```

i1      60.00000
i1      69.00000

```

## See Also

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonoct, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

# midinoteonoct

midinoteonoct — Gets a MIDI note number value as octave-point-decimal value.

## Description

*midinoteonoct* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midinoteonoct xoct, xvelocity
```

## Performance

*xoct* -- returns MIDI key translated to linear octaves during MIDI activation, remains unchanged otherwise.

*xvelocity* -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xoct* and *xvelocity* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xoct* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## Examples

Here is an example of the midinoteonoct opcode. It uses the file *midinoteonoct.csd* [examples/midinoteonoct.csd].

### Exemple 306. Example of the midinoteonoct opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midinoteonoct.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  koct init 0
  kvelocity init 0

  midinoteonoct koct, kvelocity

  ; Display the octave-point-decimal value when it changes.
  printk2 koct
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

Its output should include lines like:

```

i1      8.00000
i1      9.33333

```

## See Also

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonkey, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

# midinoteonpch

midinoteonpch — Gets a MIDI note number as a pitch-class value.

## Description

*midinoteonpch* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midinoteonpch xpch, xvelocity
```

## Performance

*xpch* -- returns MIDI key translated to octave.pch during MIDI activation, remains unchanged otherwise.

*xvelocity* -- returns MIDI velocity during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xpch* and *xvelocity* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xpch* and *xvelocity* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## Examples

Here is an example of the midinoteonpch opcode. It uses the file *midinoteonpch.csd* [examples/midinoteonpch.csd].

### Exemple 307. Example of the midinoteonpch opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime output leave only the line below:
; -o midinoteonpch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kpch init 0
  kvelocity init 0

  midinoteonpch kpch, kvelocity

  ; Display the pitch-class value when it changes.
  printk2 kpch
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

Its output should include lines like:

```

i1      8.09000
i1      9.05000

```

## See Also

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonkey, midinoteonoct, midipitchbend, midipolyaftertouch, midiprogramchange*

## Credits

Author: Michael Gogins

Example written by Kevin Conder.

New in version 4.20

# midion

midion — Generates MIDI note messages at k-rate.

## Description

Generates MIDI note messages at k-rate.

## Syntax

```
midion kchn, knum, kvel
```

## Performance

*kchn* -- MIDI channel number (1-16)

*knum* -- note number (0-127)

*kvel* -- velocity (0-127)

*midion* (k-rate note on) plays MIDI notes with current *kchn*, *knum* and *kvel*. These arguments can be varied at k-rate. Each time the MIDI converted value of any of these arguments changes, last MIDI note played by current instance of *midion* is immediately turned off and a new note with the new argument values is activated. This opcode, as well as *moscil*, can generate very complex melodic textures if controlled by complex k-rate signals.

Any number of *midion* opcodes can appear in the same Csound instrument, allowing a counterpoint-style polyphony within a single instrument.

## Examples

Here is a simple example of the *midion* opcode. It uses the file *midion\_simple.csd* [examples/midion\_simple.csd].

### Exemple 308. Simple Example of the midion opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates a minor chord over every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadac    -d        -M0   -Q1   ;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ;Triggered by MIDI notes on channel 1

    ifund notnum
    ivel  veloc

    knote1 init ifund
    knote2 init ifund + 3
    knote3 init ifund + 5

    ;minor chord on MIDI out channel 1
    ;Needs something plugged to csound's MIDI output
    midion 1, knote1,ivel
    midion 1, knote2,ivel
    midion 1, knote3,ivel

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f0 60
</CsScore>
</CsoundSynthesizer>
```

Here is another example of the midion opcode. It uses the file *midion\_scale.csd* [examples/midion\_scale.csd].

### Exemple 309. Example of the midion opcode to generate random notes from a scale.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates random notes from a given scale for every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc     -d         -M0  -Q1  ;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ; Triggered by MIDI notes on channel 1

    ivel          veloc

    krate = 8
    iscale = 100 ;f

    ; Random sequence from table f100
    krnd randh int(14),krate,-1
    knote table abs(krnd),iscale
    ; Generates random notes from the scale on ftable 100
    ; on channel 1 of csound's MIDI output
    midion 1,knote,ivel
```

```
endin  
  
</CsInstruments>  
<CsScore>  
f100 0 32 -2 40 50 60 70 80 44 54 65 74 84 39 49 69 69  
  
; Dummy ftable  
f0 60  
</CsScore>  
</CsoundSynthesizer>
```

## See Also

*moscil, midion2, noteon, noteoff, noteondur, noteondur2*

## Credits

Author: Gabriel Maldonado  
Italy  
May 1997

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# midion2

midion2 — Sends noteon and noteoff messages to the MIDI OUT port.

## Description

Sends noteon and noteoff messages to the MIDI OUT port when triggered by a value different than zero.

## Syntax

```
midion2 kchn, knum, kvel, ktrig
```

## Performance

*kchn* -- MIDI channel (1-16)

*knun* -- MIDI note number (0-127)

*kvel* -- note velocity (0-127)

*ktrig* -- trigger input signal (normally 0)

Similar to *midion*, this opcode sends noteon and noteoff messages to the MIDI out port, but only when *ktrig* is non-zero. This opcode is can work together with the output of the *trigger* opcode.

## See Also

*moscil*, *midion*, *noteon*, *noteoff*, *noteondur*, *noteondur2*

## Credits

Author: Gabriel Maldonado  
Italy  
1998

New in Csound version 3.492

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# midout

midout — Sends a generic MIDI message to the MIDI OUT port.

## Description

Sends a generic MIDI message to the MIDI OUT port.

## Syntax

```
midout kstatus, kchan, kdata1, kdata2
```

## Performance

*kstatus* -- the type of MIDI message. Can be:

- 128 (note off)
- 144 (note on)
- 160 (polyphonic aftertouch)
- 176 (control change)
- 192 (program change)
- 208 (channel aftertouch)
- 224 (pitch bend)
- 0 when no MIDI messages must be sent to the MIDI OUT port

*kchan* -- MIDI channel (1-16)

*kdata1*, *kdata2* -- message-dependent data values

*midout* has no output arguments, because it sends a message to the MIDI OUT port implicitly. It works at k-rate. It sends a MIDI message only when *kstatus* is non-zero.



### Avertissement

*Warning:* Normally *kstatus* should be set to 0. Only when the user intends to send a MIDI message, can it be set to the corresponding message type number.

## Credits

Author: Gabriel Maldonado  
Italy  
1998



New in Csound version 3.492

# midipitchbend

midipitchbend — Gets a MIDI pitchbend value.

## Description

*midipitchbend* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midipitchbend xpitchbend [, ilow] [, ihigh]
```

## Initialization

*ilow* (optional) -- optional low value after rescaling, defaults to 0.

*ihigh* (optional) -- optional high value after rescaling, defaults to 127.

## Performance

*xpitchbend* -- returns the MIDI pitch bend during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the value of *xpitchbend* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the value of *xpitchbend* remains unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## Examples

Here is an example of the midipitchbend opcode. It uses the file *midipitchbend.csd* [examples/midipitchbend.csd].

### Exemple 310. Example of the midipitchbend opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages   MIDI in
-odac         -iadc       -d             -M0   ;;RT audio I/O with MIDI in
; For Non-realtime output leave only the line below:
; -o midipitchbend.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kpb init 0

  midipitchbend kpb

  ; Display the pitch-bend value when it changes.
  printk2 kpb
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like:

```
i1      0.12695
i1      0.00000
i1     -0.01562
```

## See Also

*midchannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonkey, midinoteonoct, midinoteonpch, midipolyaftertouch, midiprogramchange*

## Credits

Author: Michael Gogins

Example written by Kevin Conder.

## New in version 4.20

# midipolyaftertouch

midipolyaftertouch — Gets a MIDI polyphonic aftertouch value.

## Description

*midipolyaftertouch* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

```
midipolyaftertouch xpolyaftertouch, xcontrollervalue [, ilow] [, ihigh]
```

## Initialization

*ilow* (optional) -- optional low value after rescaling, defaults to 0.

*ihigh* (optional) -- optional high value after rescaling, defaults to 127.

## Performance

*xpolyaftertouch* -- returns MIDI polyphonic aftertouch during MIDI activation, remains unchanged otherwise.

*xcontrollervalue* -- returns the value of the MIDI controller during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the values of *xpolyaftertouch* and *xcontrollervalue* with the corresponding values from MIDI input. If the instrument was *NOT* activated by MIDI input, the values of *xpolyaftertouch* and *xcontrollervalue* remain unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## See Also

*midichannelaftertouch*, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *mi-*

*dinoteonoct, midinoteonpch, midipitchbend, midiprogramchange*

## Credits

Author: Michael Gogins

New in version 4.20

# midiprogramchange

midiprogramchange — Gets a MIDI program change value.

## Description

*midiprogramchange* is designed to simplify writing instruments that can be used interchangeably for either score or MIDI input, and to make it easier to adapt instruments originally written for score input to work with MIDI input.

In general, it should be possible to write instrument definitions that work identically with both scores and MIDI, including both MIDI files and real-time MIDI input, without using any conditional statements, and that take full advantage of MIDI voice messages.

Note that correlating Csound instruments with MIDI channel numbers is done using the *massign* opcode for real-time performance,. For file-driven performance, instrument numbers default to MIDI channel number + 1, but the defaults are overridden by any MIDI program change messages in the file.

## Syntax

`midiprogramchange xprogram`

## Performance

*xprogram* -- returns the MIDI program change value during MIDI activation, remains unchanged otherwise.

If the instrument was activated by MIDI input, the opcode overwrites the value of *xprogram* with the corresponding value from MIDI input. If the instrument was *NOT* activated by MIDI input, the value of *xprogram* remains unchanged.

This enables score p-fields to receive MIDI input data during MIDI activation, and score values otherwise.



### Adapting a score-activated Csound instrument.

See the *MIDI interop opcodes* section for details on adapting score driven instruments for MIDI or vice-versa.

## See Also

*midichannelaftertouch*, *midicontrolchange*, *mididefault*, *midinoteoff*, *midinoteoncps*, *midinoteonkey*, *midinoteonoct*, *midinoteonpch*, *midipitchbend*, *midipolyaftertouch*

## Credits

Author: Michael Gogins

New in version 4.20

# miditempo

miditempo — Returns the current tempo at k-rate, of either the MIDI file (if available) or the score

## Description

Returns the current tempo at k-rate, of either the MIDI file (if available) or the score

## Syntax

```
ksig miditempo
```

## Credits

Author: Istvan Varga  
March 2005  
New in Csound5

# midremot

**midremot** — An opcode which can be used to implement a remote midi orchestra. This opcode will send midi events from a source machine to one destination.

## Description

With the *midremot* and *midglobal* opcodes you are able to perform instruments on remote machines and control them from a master machine. The remote opcodes are implemented using the master/client model. All the machines involved contain the same orchestra but only the master machine contains the information of the midi score. During the performance the master machine sends the midi events to the clients. The *midremot* opcode will send events from a source machine to one destination if you want to send events to many destinations (broadcast) use the *midglobal* opcode instead. These two opcodes can be used in combination.

## Syntax

```
midremotdestination, isource, instrnum [,instrnum...]
```

## Initialization

*idestination* -- a string that is the intended host computer (e.g. 192.168.0.100). This is the destination host which receives the events from the given instrument.

*isource* -- a string that is the intended host computer (e.g. 192.168.0.100). This is the source host which generates the events of the given instrument and sends it to the address given by *idestination*.

*instrnum* -- a list of instrument numbers which will be played on the destination machine

## Example

## Examples

Here is an example of the *midremot* opcode. It uses the files *insremot.csd* [examples/midremot.csd].

### Exemple 311. Example of the *insremot* opcode.

The example shows a Bach fugue played on 4 remote computers. The master machine is named "192.168.1.100", client1 "192.168.1.101" and so on. Start the clients on each machine (they will be waiting to receive the events from the master machine) and then start the master. Here is the command on linux to start a client (`csound -dm0 -odac -+rtaudio=alsa midremot.csd -+rtmidi=Null`), and the command on the master machine will look like this (`csound -dm0 -odac -+rtaudio=alsa midremot.csd -F midremot.mid`).

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```



```

; Audio out      Audio in
-odac            -iadc            ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o midremot.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 441
ksmps = 100
nchnls = 2

massign 1,1
massign 2,2
massign 3,3
massign 4,4
massign 5,5

gal init 0
ga2 init 0

gil sfload      "19Trumpet.sf2"

gi2 sfload      "01hpschd.sf2"

gi3 sfload      "07AcousticGuitar.sf2"

gi4 sfload      "22Bassoon.sf2"

gitab ftgen 1,0,1024,10,1

midremot "192.168.1.100", "192.168.1.101", 1
midremot "192.168.1.100", "192.168.1.102", 2
midremot "192.168.1.100", "192.168.1.103", 3

midglobal "192.168.1.100", 5

        instr 1
        sfpassign 0, gil
ifreq cpsmidi
iamp ampmidi 10
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
      outs a1,a2
vincr ga1, a1*.5
vincr ga2, a2*.5
      endin

        instr 2
        sfpassign 0,    gi2
ifreq cpsmidi
iamp ampmidi 15
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
      outs a1,a2
vincr ga1, a1*.4
vincr ga2, a2*.4
      endin

        instr 3
        sfpassign 0,    gi3
ifreq cpsmidi
iamp ampmidi 10
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
      outs a1,a2
vincr ga1, a1*.5
vincr ga2, a2*.5
      endin

        instr 4
        sfpassign 0,    gi4

```

```

ifreq cpsmidi
iamp ampmidi 15
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
      outs a1,a2
vincr ga1, a1*.5
vincr ga2, a2*.5
      endin

instr 5
  kamp midic7 1,0,1
  denorm ga1
  denorm ga2
aL, aR reverbsc ga1, ga2, .9, 16000, sr, 0.5
      outs aL, aR
      ga1 = 0
      ga2 = 0
      endin

</CsInstruments>
<CsScore>
; Score
f0 160
</CsScore>
</CsoundSynthesizer>

```

## See also

*insglobal, insremot, midglobal, remoteport*

## Credits

Author: Simon Schampijer  
2006

New in version 5.03

# midglobal

**midglobal** — An opcode which can be used to implement a remote midi orchestra. This opcode will broadcast the midi events to all the machines involved in the remote concert.

## Description

With the *midremot* and *midglobal* opcodes you are able to perform instruments on remote machines and control them from a master machine. The remote opcodes are implemented using the master/client model. All the machines involved contain the same orchestra but only the master machine contains the information of the midi score. During the performance the master machine sends the midi events to the clients. The *midglobal* opcode sends the events to all the machines involved in the remote concert. These machines are determined by the *midremot* definitions made above the *midglobal* command. To send events to only one machine use *midremot*.

## Syntax

```
midglobal source, instrnum [,instrnum...]
```

## Initialization

*source* -- a string that is the intended host computer (e.g. 192.168.0.100). This is the source host which generates the events of the given instrument(s) and sends it to all the machines involved in the remote concert.

*instrnum* -- a list of instrument numbers which will be played on the destination machines

## Examples

See the entry for *midremot* for an example of usage.

## See also

*insglobal, insremot, midremot, remoteport*

## Credits

Author: Simon Schampijer  
2006

New in version 5.03

# min

min — Produces a signal that is the minimum of any number of input signals.

## Description

The *min* opcode takes any number of a-rate or k-rate signals as input (all of the same rate), and outputs a signal at the same rate that is the minimum of all of the inputs. For a-rate signals, the inputs are compared one sample at a time (i.e. *min* does not scan an entire ksmps period of a signal for its local minimum as the *max\_k* opcode does).

## Syntax

```
amin min ain1 [, ain2] [, ain3] [, ain4] [...]
```

```
kmin min kin1 [, kin2] [, kin3] [, kin4] [...]
```

## Performance

*ain1, ain2, ...* -- a-rate signals to be compared.

*kin1, kin2, ...* -- k-rate signals to be compared.

## See Also

*max, maxabs, minabs, maxaccum, minaccum, maxabsaccum, minabsaccum, max\_k*

## Credits

Author: Anthony Kozar  
March 2006

New in Csound version 5.01

# minabs

minabs — Produces a signal that is the minimum of the absolute values of any number of input signals.

## Description

The *minabs* opcode takes any number of a-rate or k-rate signals as input (all of the same rate), and outputs a signal at the same rate that is the minimum of all of the inputs. It is identical to the *min* opcode except that it takes the absolute value of each input before comparing them. Therefore, the output is always non-negative. For a-rate signals, the inputs are compared one sample at a time (i.e. *minabs* does not scan an entire ksmpls period of a signal for its local minimum as the *max\_k* opcode does).

## Syntax

```
amin minabs ain1 [, ain2] [, ain3] [, ain4] [...]
```

```
kmin minabs kin1 [, kin2] [, kin3] [, kin4] [...]
```

## Performance

*ain1, ain2, ...* -- a-rate signals to be compared.

*kin1, kin2, ...* -- k-rate signals to be compared.

## See Also

*maxabs, max, min, maxaccum, minaccum, maxabsaccum, minabsaccum, max\_k*

## Credits

Author: Anthony Kozar  
March 2006

New in Csound version 5.01

# minabsaccum

minabsaccum — Accumulates the minimum of the absolute values of audio signals.

## Description

*minabsaccum* compares two audio-rate variables and stores the minimum of their absolute values into the first.

## Syntax

```
minabsaccum aAccumulator, aInput
```

## Performance

*aAccumulator* -- audio variable to store the minimum value

*aInput* -- signal that *aAccumulator* is compared to

The *minabsaccum* opcode is designed to accumulate the minimum value from among many audio signals that may be in different note instances, different channels, or otherwise cannot all be compared at once using the *minabs* opcode. *minabsaccum* is identical to *minaccum* except that it takes the absolute value of *aInput* before the comparison. Its semantics are similar to *vincr* since *aAccumulator* is used as both an input and an output variable, except that *minabsaccum* keeps the minimum absolute value instead of adding the signals together. *minabsaccum* performs the following operation on each pair of samples:

$$\text{if } (\text{abs}(\text{aInput}) < \text{aAccumulator}) \text{ aAccumulator} = \text{abs}(\text{aInput})$$

*aAccumulator* will usually be a global audio variable. At the end of any given computation cycle (k-period), after its value is read and used in some way, the accumulator variable should usually be reset to some large enough positive value that will always be greater than the input signals to which it is compared.

## See Also

*maxabsaccum*, *maxaccum*, *minaccum*, *max*, *min*, *maxabs*, *minabs*, *vincr*

## Credits

Author: Anthony Kozar  
March 2006

New in Csound version 5.01

# minaccum

minaccum — Accumulates the minimum value of audio signals.

## Description

*minaccum* compares two audio-rate variables and stores the minimum value between them into the first.

## Syntax

```
minaccum aAccumulator, aInput
```

## Performance

*aAccumulator* -- audio variable to store the minimum value

*aInput* -- signal that *aAccumulator* is compared to

The *minaccum* opcode is designed to accumulate the minimum value from among many audio signals that may be in different note instances, different channels, or otherwise cannot all be compared at once using the *min* opcode. Its semantics are similar to *vincr* since *aAccumulator* is used as both an input and an output variable, except that *minaccum* keeps the minimum value instead of adding the signals together. *minaccum* performs the following operation on each pair of samples:

```
if (aInput < aAccumulator) aAccumulator = aInput
```

*aAccumulator* will usually be a global audio variable. At the end of any given computation cycle (k-period), after its value is read and used in some way, the accumulator variable should usually be reset to some large enough positive value that will always be greater than the input signals to which it is compared.

## See Also

*maxaccum*, *maxabsaccum*, *minabsaccum*, *max*, *min*, *maxabs*, *minabs*, *vincr*

## Credits

Author: Anthony Kozar  
March 2006

New in Csound version 5.01

# mirror

mirror — Reflects the signal that exceeds the low and high thresholds.

## Description

Reflects the signal that exceeds the low and high thresholds.

## Syntax

```
ares mirror asig, klow, khigh
```

```
ires mirror isig, ilow, ihigh
```

```
kres mirror ksig, klow, khigh
```

## Initialization

*isig* -- input signal

*ilow* -- low threshold

*ihigh* -- high threshold

## Performance

*xsig* -- input signal

*klow* -- low threshold

*khigh* -- high threshold

*mirror* « reflects » the signal that exceeds the low and high thresholds.

This opcode is useful in several situations, such as table indexing or for clipping and modeling a-rate, i-rate or k-rate signals.

## See Also

*limit*, *wrap*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.49



# MixerSetLevel

MixerSetLevel — Sets the level of a send to a buss.

## Syntax

```
MixerSetLevel isend, ibuss, kgain
```

## Description

Sets the level at which signals from the send are added to the buss. The actual sending of the signal to the buss is performed by the *MixerSend* opcode.

## Initialization

*isend* -- The number of the send, for example the number of the instrument sending the signal (but any integer can be used).

*ibuss* -- The number of the buss, for example the number of the instrument receiving the signal (but any integer can be used).

Setting the gain for a buss also creates the buss.

## Performance

*kgain* -- The level (any real number) at which the signal from the send will be mixed onto the buss. The default is 0.

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses before the next kperiod.

## Examples

In the orchestra, define an instrument to control mixer levels:

```
instr 1
  MixerSetLevel      p4, p5, p6
endin
```

In the score, use that instrument to set mixer levels:

```
; SoundFonts
; to Chorus
i 1 0 0 100 200 0.9
; to Reverb
i 1 0 0 100 210 0.7
; to Output
i 1 0 0 100 220 0.3

; Kelley Harpsichord
; to Chorus
i 1 0 0 3 200 0.30
; to Reverb
```

```
i 1 0 0 3 210 0.9
; to Output
i 1 0 0 3 220 0.1

; Chorus to Reverb
i 1 0 0 200 210 0.5
; Chorus to Output
i 1 0 0 200 220 0.5
; Reverb to Output
i 1 0 0 210 220 0.2
```

## Credits

Michael Gogins (gogins at pipeline dot com).

# MixerSetLevel\_i

MixerSetLevel\_i — Sets the level of a send to a buss.

## Syntax

```
MixerSetLevel_i isend, ibuss, igain
```

## Description

Sets the level at which signals from the send are added to the buss. This opcode, because all parameters are irate, may be used in the orchestra header. The actual sending of the signal to the buss is performed by the *MixerSend* opcode.

## Initialization

*isend* -- The number of the send, for example the number of the instrument sending the signal (but any integer can be used).

*ibuss* -- The number of the buss, for example the number of the instrument receiving the signal (but any integer can be used).

*igain* -- The level (any real number) at which the signal from the send will be mixed onto the buss. The default is 0.

Setting the gain for a buss also creates the buss.

## Performance

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses before the next kperiod.

## Examples

In the orchestra header, set the gain for the send from buss 3 to buss 4:

```
MixerSetLevel_i      3, 4, 0.76
```

## Credits

Michael Gogins (gogins at pipeline dot com).

# MixerGetLevel

MixerGetLevel — Gets the level of a send to a buss.

## Syntax

```
kgain MixerGetLevel isend, ibuss
```

## Description

Gets the level at which signals from the send are being added to the buss. The actual sending of the signal to the buss is performed by the *MixerSend* opcode.

## Initialization

*isend* -- The number of the send, for example the number of the instrument sending the signal.

*ibuss* -- The number of the buss, for example the number of the instrument receiving the signal.

## Performance

*kgain* -- The level (any real number) at which the signal from the send will be mixed onto the buss.

This opcode reports the level set by *MixerSetLevel* for a send and buss pair.

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses to 0 before the next kperiod.

## Credits

Michael Gogins (gogins at pipeline dot com).

# MixerSend

MixerSend — Mixes an arate signal into a channel of a buss.

## Syntax

```
MixerSend asignal, isend, ibuss, ichannel
```

## Description

Mixes an arate signal into a channel of a buss.

## Initialization

*isend* -- The number of the send, for example the number of the instrument sending the signal. The gain of the send is controlled by the *MixerSetLevel* opcode. The reason that the sends are numbered is to enable different levels for different sends to be set independently of the actual level of the signals.

*ibuss* -- The number of the buss, for example the number of the instrument receiving the signal.

*ichannel* -- The number of the channel. Each buss has `nchnls` channels.

## Performance

*asignal* -- The signal that will be mixed into the indicated channel of the buss.

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses to 0 before the next kperiod.

## Examples

```
instr 100 ; Fluidsynth output
; INITIALIZATION
; Normalize so iamplitude for p5 of 80 == ampdb(80).
iamplitude      =      ampdb(p5) * 2.0
; AUDIO
aleft, aright    fluidAllOut      giFluidsynth
asig1            =      aleft * iamplitude
asig2            =      aright * iamplitude

; To the chorus.
MixerSend        asig1, 100, 200, 0
MixerSend        asig2, 100, 200, 1
; To the reverb.
MixerSend        asig1, 100, 210, 0
MixerSend        asig2, 100, 210, 1
; To the output.
MixerSend        asig1, 100, 220, 0
MixerSend        asig2, 100, 220, 1

endin
```

## Credits

Michael Gogins (gogins at pipeline dot com).

# MixerReceive

MixerReceive — Receives an arate signal from a channel of a buss.

## Syntax

```
asignal MixerReceive ibuss, ichannel
```

## Description

Receives an arate signal that has been mixed onto a channel of a buss.

## Initialization

*ibuss* -- The number of the buss, for example the number of the instrument receiving the signal.

*ichannel* -- The number of the channel. Each buss has `nchnls` channels.

## Performance

*asignal* -- The signal that has been mixed onto the indicated channel of the buss.

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses to 0 before the next kperiod.

## Examples

```
instr 220 ; Master output
    ; It applies a bass enhancement, compression and fadeout
    ; to the whole piece, outputs signals, and clears the mixer.
    a1 MixerReceive 220, 0
    a2 MixerReceive 220, 1
    ; Bass enhancement
    a11 butterlp a1, 100
    a12 butterlp a2, 100
    a1 = a11*1.5 +a1
    a2 = a12*1.5 +a2

    ; Global amplitude shape
    kenv linseg 0., p5 / 2.0, p4, p3 - p5, p4, p5 / 2.0, 0.
    a1=a1*kenv
    a2=a2*kenv

    ; Compression
    a1 dam a1, 5000, 0.5, 1, 0.2, 0.1
    a2 dam a2, 5000, 0.5, 1, 0.2, 0.1

    ; Remove DC bias
    a1blocked dcblock a1
    a2blocked dcblock a2

    ; Output signals
    outs a1blocked, a2blocked
    MixerClear
endin
```

## Credits

Michael Gogins (gogins at pipeline dot com).

# MixerClear

MixerClear — Resets all channels of a buss to 0.

## Syntax

**MixerClear**

## Description

Resets all channels of a buss to 0.

## Performance

Use of the mixer requires that instruments setting gains have smaller numbers than instruments sending signals, and that instruments sending signals have smaller numbers than instruments receiving those signals. However, an instrument may have any number of sends or receives. After the final signal is received, *MixerClear* must be invoked to reset the busses to 0 before the next kperiod.

## Examples

```
instr 220 ; Master output
    ; It applies a bass enhancement, compression and fadeout
    ; to the whole piece, outputs signals, and clears the mixer.
    a1 MixerReceive 220, 0
    a2 MixerReceive 220, 1
    ; Bass enhancement
    a11 butterlp a1, 100
    a12 butterlp a2, 100
    a1 = a11*1.5 +a1
    a2 = a12*1.5 +a2

    ; Global amplitude shape
    kenv linseg 0., p5 / 2.0, p4, p3 - p5, p4, p5 / 2.0, 0.
    a1=a1*kenv
    a2=a2*kenv

    ; Compression
    a1 dam a1, 5000, 0.5, 1, 0.2, 0.1
    a2 dam a2, 5000, 0.5, 1, 0.2, 0.1

    ; Remove DC bias
    a1blocked dcblock a1
    a2blocked dcblock a2

    ; Output signals
    outs a1blocked, a2blocked
    MixerClear
endin
```

## Credits

Author: Michael Gogins (gogins at pipeline dot com).



# mode

mode — A filter that simulates a mass-spring-damper system

## Description

Filters the incoming signal with the specified resonance frequency and quality factor. It can also be seen as a signal generator for high quality factor, with an impulse for the excitation. You can combine several modes to build complex instruments such as bells or guitar tables.

## Syntax

```
aout mode ain, kfreq, kQ [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- if non zero skip the initialisation of the filter.

## Performance

*aout* -- filtered signal

*ain* -- signal to filter

*kfreq* -- resonant frequency of the filter



### Warning

This filter becomes unstable if  $sr/ikfreq < \pi$  (e.g  $ikfreq > 14037$  Hz @44kHz)

*kQ* -- quality factor of the filter

The resonance time is roughly proportionnal to  $kQ/kfreq$ .

See *Modal Frequency Ratios* for frequency ratios of real intruments which can be used to determine the values of *kfreq*.

## Exampless

Here is an example of the mode opcode. It uses the file *mode.csd* [examples/mode.csd].

### Exemple 312. Example of the mode opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
```

```

-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o moogvcf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1; 2 modes excitator

idur init p3
ifreql1 init p4
ifreql2 init p5
iQl1    init p6
iQl2    init p7
iamp    init ampdb(p8)
ifreq21 init p9
ifreq22 init p10
iQ21    init p11
iQ22    init p12

; to simulate the shock between the excitator and the resonator
ashock mpulse 3,0

aexc1 mode ashock,ifreql1,iQl1
aexc1 = aexc1*iamp
aexc2 mode ashock,ifreql2,iQl2
aexc2 = aexc2*iamp

aexc = (aexc1+aexc2)/2

;"Contact" condition : when aexc reaches 0, the excitator looses
;contact with the resonator, and stops "pushing it"
aexc limit aexc,0,3*iamp

; 2modes resonator

ares1 mode aexc,ifreq21,iQ21
ares2 mode aexc,ifreq22,iQ22

ares = (ares1+ares2)/2

display aexc+ares,p3
outs aexc+ares,aexc+ares

endin

</CsInstruments>
<CsScore>

;wooden excitator against glass resonator
i1 0 8 1000 3000 12 8 70 440 888 500 420

;felt against glass
i1 4 8 80 188 8 3 70 440 888 500 420

;wood against wood
i1 8 8 1000 3000 12 8 70 440 630 60 53

;felt against wood
i1 12 8 80 180 8 3 70 440 630 60 53

i1 16 8 1000 3000 12 8 70 440 888 2000 1630
i1 23 8 80 180 8 3 70 440 888 2000 1630

;With a metallic excitator

i1 33 8 1000 1800 1000 720 70 440 882 500 500
i1 37 8 1000 1800 1000 850 70 440 630 60 53

i1 42 8 1000 1800 2000 1720 70 440 442 500 500

```

```
</CsScore>  
</CsoundSynthesizer>
```

## Credits

Original UDO and documentation/example by François Blanc

Opcode translation to C-code by Steven Yi

New in version 5.04

# modmatrix

modmatrix — Opcode matrice de modulation avec optimisation pour les matrices creuses.

## Description

On peut utiliser cet opcode pour faire moduler un grand nombre de paramètres variables de taux-k par un grand nombre de variables modulantes de taux-k, avec une pondération arbitraire de chaque connexion paramètre-modulateur. Des ftables de Csound sont utilisées pour contenir les variables en entrée (les paramètres), les variables modulantes et les coefficients de pondération. Les variables de sorties sont écrites dans une autre ftable de Csound.

## Syntaxe

```
modmatrix iresfn, isrcmodfn, isrcparmf, imodscale, inum_mod, \\  
inum_parm, kupdate
```

## Initialisation

*iresfn* -- numéro de la ftable pour les variables de sortie.

*isrcmodfn* -- numéro de la ftable pour les variables sources de modulation.

*isrcparmf* -- numéro de la ftable pour les paramètres variables en entrée.

*imodscale* -- matrice des coefficients de pondération/routage. C'est aussi une ftable de Csound, utilisée comme une matrice de *inum\_mod* lignes et *inum\_parm* colonnes.

*inum\_mod* -- nombre de variables de modulation.

*inum\_parm* -- nombre de paramètres variables (en entrée et en sortie).

Les arguments *inum\_mod* et *inum\_parm* ne doivent pas nécessairement être des puissances de deux.

## Exécution

*kupdate* -- indicateur pour la mise à jour des coefficients de pondération. Quand l'indicateur a une valeur non nulle, les coefficients de pondération sont lus directement de la table *imodscale*. Quand l'indicateur vaut zéro, les coefficients de pondérations sont parcourus et une matrice de pondération optimisée est stockée en interne dans l'opcode.

Chaque modulateur dans *isrcmodfn*, est pondéré par un coefficient (dans *imodscale*) déterminant son degré d'influence sur chaque paramètre. Puis tous les modulateurs pour un paramètre sont additionnés et la valeur de modulation résultante est ajoutée à la valeur du paramètre d'entrée lu dans *isrcparmf*. Enfin, les valeurs du paramètre de sortie sont écrites dans la table *iresfn*.

Les tables suivantes donnent un aperçu du processus exécuté par l'opcode *modmatrix*, dans un exemple simplifié utilisant 3 paramètres et 2 modulateurs. Appelons les paramètres "cps1", "cps2" et "cutoff", et les modulateurs "lfo1" et "lfo2".

Les variables d'entrée peuvent avoir ces valeurs à un certain moment :

**Tableau 13.**

	<b>cps1</b>	<b>cps2</b>	<b>cutoff</b>
<i>isrcparmfn</i>	400	800	3

... tandis que les variables de modulation ont ces valeurs :

**Tableau 14.**

	<b>lfo1</b>	<b>lfo2</b>
<i>isrcmodfn</i>	0.5	-0.2

Les coefficients de pondération/routage sont :

**Tableau 15.**

<i>imodscale</i>	<b>cps1</b>	<b>cps2</b>	<b>cutoff</b>
<i>lfo1</i>	40	0	-2
<i>lfo2</i>	-50	100	3

... et les valeurs de sortie résultantes sont :

**Tableau 16.**

	<b>cps1</b>	<b>cps2</b>	<b>cutoff</b>
<i>iresfn</i>	430	780	1.4
<i>lfo2</i>	-50	100	3

La valeur de sortie pour "cps1" est calculée comme  $400 + (0.5 * 40) + (-0.2 * -50)$ , de même pour "cps2"  $800 + (0.5 * 0) + (-0.2 * 100)$ , et pour "cutoff" :  $3 + (0.5 * -2) + (-0.2 * 3)$

La ftable *imodscale* peut être spécifiée dans la partition comme ceci :

```
f1 0 8 -2 200 0 2 50 300 -1.5
```

Ou mieux en utilisant *ftgen* dans l'orchestre :

```
gimodscale ftgen 0, 0, 8, -2, 200, 0, 2, 50, 300, -1.5
```

Evidemment, les paramètres variables et les modulateurs n'ont pas nécessairement des valeurs statiques, de même que la table des coefficients de pondération/routage peut être continuellement renouvelée au moyen d'opcodes comme *tablew*.

## Exemples

Voici un exemple de l'opcode `modmatrix`. Il utilise le fichier `modmatrix.csd` [examples/modmatrix.csd].

### Exemple 313. Example of the `modmatrix` opcode.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio flags here according to platform
; Audio out   Audio in
;-odac        -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
-o modmatrix.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

    sr = 44100
    kr = 441
    ksmpls = 100
    nchnls = 2
    odbfs = 1

; basic waveforms
giSine ftgen 0, 0, 65537, 10, 1 ; sine wave
giSaw   ftgen 0, 0, 4097, 7, 1, 4096, -1 ; saw (linear)
giSoftSaw ftgen 0, 0, 65537, 30, giSaw, 1, 10 ; soft saw (only 10 first harmonics)

; modmatrix tables
giMaxNumParam = 128
giMaxNumMod = 32
giParam_In ftgen 0, 0, giMaxNumParam, 2, 0 ; input parameters table
; output parameters table (parameter values with added modulators)
giParam_Out ftgen 0, 0, giMaxNumParam, 2, 0
giModulators ftgen 0, 0, giMaxNumMod, 2, 0 ; modulators table
; modulation scaling and routing (mod matrix) table, start with empty table
giModScale ftgen 0, 0, giMaxNumParam*giMaxNumMod, -2, 0

;*****
; generate the modulator signals
;*****
instr 1

; LFO1, 1.5 Hz, normalized range (0.0 to 1.0)
kLFO1 oscil 0.5, 1.5, giSine ; generate LFO signal
kLFO1 = kLFO1+0.5 ; offset

; LFO2, 0.4 Hz, normalized range (0.0 to 1.0)
kLFO2 oscil 0.5, 0.4, giSine ; generate LFO signal
kLFO2 = kLFO2+0.5 ; offset

; write modulators to table
tablew kLFO1, 0, giModulators
tablew kLFO2, 1, giModulators

endin

;*****
; set parameter values
;*****
instr 2

; Here we can set the parameter values
icps1 = p4
icps2 = p5
icutoff = p6

; write parameters to table
tableiw icps1, 0, giParam_In
tableiw icps2, 1, giParam_In
tableiw icutoff, 2, giParam_In

endin
```

```

;*****
; mod matrix edit
;*****
    instr 3

; Here we can write to the modmatrix table by using tablew or tableiw

iLfo1ToCps1 = p4
iLfo1ToCps2 = p5
iLfo1ToCutoff = p6
iLfo2ToCps1 = p7
iLfo2ToCps2 = p8
iLfo2ToCutoff = p9

    tableiw iLfo1ToCps1, 0, giModScale
    tableiw iLfo1ToCps2, 1, giModScale
    tableiw iLfo1ToCutoff, 2, giModScale
    tableiw iLfo2ToCps1, 3, giModScale
    tableiw iLfo2ToCps2, 4, giModScale
    tableiw iLfo2ToCutoff, 5, giModScale

; and set the update flag for modulator matrix
; ***(must update to enable changes)
ktrig init 1
    chnset ktrig, "modulatorUpdateFlag"
ktrig = 0

    endin

;*****
; mod matrix
;*****
    instr 4

; get the update flag
kupdate chnget "modulatorUpdateFlag"

; run the mod matrix
inum_mod = 2
inum_parm = 3
    modmatrix giParam_Out, giModulators, giParam_In, \
    giModScale, inum_mod, inum_parm, kupdate

; and reset the update flag
    chnset 0, "modulatorUpdateFlag" ; reset the update flag

    endin

;*****
; audio generator to test values
;*****
    instr 5

; basic parameters
iamp = ampdbfs(-5)

; read modulated parameters from table
kcps1 table 0, giParam_Out
kcps2 table 1, giParam_Out
kcutoff table 2, giParam_Out

; set filter parameters
kCF_freq1 = kcps1*kcutoff
kCF_freq2 = kcps2*kcutoff
kReso = 0.7
kDist = 0.3

; oscillators and filters
a1 oscili iamp, kcps1, giSoftSaw
a1 lpf18 a1, kCF_freq1, kReso, kDist

a2 oscili iamp, kcps2, giSoftSaw
a2 lpf18 a2, kCF_freq2, kReso, kDist

    outs a1, a2

    endin

</CsInstruments>
<CsScore>

```

```

;*****
; set initial parameters
; cps1 cps2 cutoff
i2 0 1 400 800 3

;*****
; set modmatrix values
; lfo1ToCps1 lfo1ToCps2 lfo1ToCut lfo2ToCps1 lfo2ToCps2 lfo2ToCut
i3 0 1          40          0          -2          -50          100          3

;*****
; start "always on" instruments
#define SCORELEN # 20 # ; set length of score

i1 0 $SCORELEN ; start modulators
i4 0 $SCORELEN ; start mod matrix
i5 0 $SCORELEN ; start audio oscillator

e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*Opcodes d'Algèbre Linéaire, Opcodes Vectoriels, tablew.*

## Crédits

Auteurs : Oeyvind Brandtsegg et Thom Johansen

Nouveau dans la version 5.12



# monitor

monitor — Retourne la trame audio de spout.

## Description

Retourne la trame audio de spout (s'il est actif), sinon retourne zéro.

## Syntaxe

```
aout1 [,aout2 ... aoutX] monitor
```

## Exécution

Cet opcode peut être utilisé pour surveiller le signal de sortie de Csound. Il ne faut pas l'utiliser pour un traitement en aval du signal.

Voir l'article sur l'opcode *fout* pour un exemple de l'utilisation de *monitor*.

## Voir Aussi

*fout*, les *opcodes Mixer* et le *Système de Patch Zak*.

## Crédits

Istvan Varga 2006

# moog

moog — Emulation d'un synthétiseur mini-Moog.

## Description

Emulation d'un synthétiseur mini-Moog.

## Syntaxe

ares **moog** kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn

## Initialisation

*iafn*, *iwfn*, *ivfn* -- les trois numéros des tables contenant la forme d'onde de l'attaque (non bouclée), la forme d'onde de la boucle principale, et la forme d'onde du vibrato. Les fichiers *mandpluk.aiff* [exemples/mandpluk.aiff] et *impuls20.aiff* [exemples/impuls20.aiff] conviennent bien pour les deux premières et une sinusoïde fera l'affaire pour la troisième.



### Note

Les fichiers « *mandpluk.aiff* » et « *impuls20.aiff* » sont aussi disponibles à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- amplitude de la note.

*kfreq* -- fréquence de la note.

*kfiltq* -- Q du filtre, compris entre 0,8 et 0,9

*kfiltrate* -- taux de contrôle pour le filtre, compris entre 0 et 0,0002

*kvibf* -- fréquence du vibrato en Hertz. L'intervalle conseillé va de 0 à 12

*kvamp* -- amplitude du vibrato

## Exemples

Voici un exemple de l'opcode moog. Il utilise les fichiers *moog.csd* [exemples/moog.csd], *mandpluk.aiff* [exemples/mandpluk.aiff] et *impuls20.aiff* [exemples/impuls20.aiff].

### Exemple 314. Exemple de l'opcode moog.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

<CsoundSynthesizer>

```

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o moog.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kfreq = 220
  kfiltq = 0.81
  kfiltrate = 0
  kvibf = 1.4
  kvamp = 2.22
  iafn = 1
  iwfn = 2
  ivfn = 3

  am moog kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn

; It tends to get loud, so clip moog's amplitude at 30,000.
al clip am, 2, 30000
out al
endin

</CsInstruments>
<CsScore>

; Table #1: the "mandpluk.aiff" audio file
f 1 0 8192 1 "mandpluk.aiff" 0 0 0
; Table #2: the "impuls20.aiff" audio file
f 2 0 256 1 "impuls20.aiff" 0 0 0
; Table #3: a sine wave
f 3 0 256 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsSoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# moogladder

moogladder — Moog ladder lowpass filter.

## Description

Moogladder is an new digital implementation of the Moog ladder filter based on the work of Antti Huovilainen, described in the paper "Non-Linear Digital Implementation of the Moog Ladder Filter" (Proceedings of DaFX04, Univ of Napoli). This implementation is probably a more accurate digital representation of the original analogue filter.

## Syntax

```
asig moogladder ain, kcf, kres[, istor]
```

## Initialization

*istor* --initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*asig* -- input signal.

*kcf* -- filter cutoff frequency

*kres* -- resonance, generally  $< 1$ , but not limited to it. Higher than 1 resonance values might cause aliasing, analogue synths generally allow resonances to be above 1.

## Examples

### Exemple 315. Example

```
kfe      expseg 500, p3*0.9, 1800, p3*0.1, 3000
kenv     linen 10000, 0.05, p3, 0.05
asig     buzz  kenv, 100, sr/(200), 1
afil     moogladder asig, kfe, 1

        out afil
```

## Credits

Author: Victor Lazzarini  
January 2005

New plugin in version 5

January 2005.

# moogvcf

moogvcf — A digital emulation of the Moog diode ladder filter configuration.

## Description

A digital emulation of the Moog diode ladder filter configuration.

## Syntax

```
ares moogvcf asig, xfco, xres [,iscale, iskip]
```

## Initialization

*iscale* (optional, default=1) -- internal scaling factor. Use if *asig* is not in the range +/-1. Input is first divided by *iscale*, then output is multiplied *iscale*. Default value is 1. (New in Csound version 3.50)

*iskip* (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

## Performance

*asig* -- input signal

*xfco* -- filter cut-off frequency in Hz. As of version 3.50, may i-,k-, or a-rate.

*xres* -- amount of resonance. Self-oscillation occurs when *xres* is approximately one. As of version 3.50, may a-rate, i-rate, or k-rate.

*moogvcf* is a digital emulation of the Moog diode ladder filter configuration. This emulation is based loosely on the paper « Analyzing the Moog VCF with Considerations for Digital Implementation » by Stilson and Smith (CCRMA). This version was originally coded in Csound by Josep Comajuncosas. Some modifications and conversion to C were done by Hans Mikelson



### Avertissement

This filter requires that the input signal be normalized to one. This can be easily achieved using *Odbfs*, like this:

```
ares moogvcf asig, kfco, kres, Odbfs
```

You can also use *moogvcf2* which defaults scaling to *Odbfs*.

## Examples

Here is an example of the moogvcf opcode. It uses the file *moogvcf.csd* [examples/moogvcf.csd].

### Exemple 316. Example of the moogvcf opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d             ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o moogvcf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a nice sawtooth waveform.
asig vco 32000, 220, 1

; Vary the filter-cutoff frequency from .2 to 2 KHz.
kfco line 200, p3, 2000

; Set the resonance amount to one.
krez init 1

; Scale the amplitude to 32768.
iscale = 32768

al moogvcf asig, kfco, krez, iscale

out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*moogvcf2, biquad, rezzv*

## Credits

Author: Hans Mikelson  
October 1998

Example written by Kevin Conder.

New in Csound version 3.49

# moogvcf2

moogvcf2 — A digital emulation of the Moog diode ladder filter configuration.

## Description

A digital emulation of the Moog diode ladder filter configuration.

## Syntax

```
ares moogvcf2 asig, xfco, xres [,iscale, iskip]
```

## Initialization

*iscale* (optional, default=0dBfs) -- internal scaling factor, as the operation of the code requires the signal to be in the range +/-1. Input is first divided by *iscale*, then output is multiplied by *iscale*.

*iskip* (optional, default=0) -- if non zero skip the initialisation of the filter.

## Performance

*asig* -- input signal

*xfco* -- filter cut-off frequency in Hz. which may be i-,k-, or a-rate.

*xres* -- amount of resonance. Self-oscillation occurs when *xres* is approximately one. May be a-rate, i-rate, or k-rate.

*moogvcf2* is a digital emulation of the Moog diode ladder filter configuration. This emulation is based loosely on the paper « Analyzing the Moog VCF with Considerations for Digital Implementation » by Stilson and Smith (CCRMA). This version was originally coded in Csound by Josep Comajuncosas. Some modifications and conversion to C were done by Hans Mikelson and then adjusted.

*moogvcf2* is identical to *moogvcf*, except that the *iscale* parameter defaults to *0dbfs* instead of 0, guaranteeing that amplitude will usually be OK.

## Examples

Here is an example of the moogvcf2 opcode. It uses the file *moogvcf2.csd* [examples/moogvcf2.csd].

### Exemple 317. Example of the moogvcf2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o moogvcf.wav -W ;; for file output any platform
```



```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a nice sawtooth waveform.
asig vco 32000, 220, 1

; Vary the filter-cutoff frequency from .2 to 2 KHz.
kfco line 200, p3, 2000

; Set the resonance amount to one.
krez init 1

al moogvcf2 asig, kfco, krez

out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>

```

## See Also

*moogvcf, biquad, rezy*

## Credits

Author: Hans Mikelson and John ffitch  
 October 1998/ July 2006

Example written by Kevin Conder.

New in Csound version 5.03

# moscil

moscil — Sends a stream of the MIDI notes.

## Description

Sends a stream of the MIDI notes.

## Syntax

```
moscil kchn, knum, kvel, kdur, kpause
```

## Performance

*kchn* -- MIDI channel number (1-16)

*knum* -- note number (0-127)

*kvel* -- velocity (0-127)

*kdur* -- note duration in seconds

*kpause* -- pause duration after each noteoff and before new note in seconds

*moscil* and *midion* are the most powerful MIDI OUT opcodes. *moscil* (MIDI oscil) plays a stream of notes of *kdur* duration. Channel, pitch, velocity, duration and pause can be controlled at k-rate, allowing very complex algorithmically generated melodic lines. When current instrument is deactivated, the note played by current instance of *moscil* is forcedly truncated.

Any number of *moscil* opcodes can appear in the same Csound instrument, allowing a counterpoint-style polyphony within a single instrument.

## Examples

Here is an example of the *moscil* opcode. It uses the file *moscil.csd* [examples/moscil.csd].

### Exemple 318. Example of the moscil opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates a stream of notes for every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc     -d         -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ;Triggered by MIDI notes on channel 1

  inote notnum
  ivel      veloc

  kpitch = 40
  kfreq  = 2

  kdur   = .04
  kpause = .1

  k1      lfo      kpitch, kfreq,5

  ;plays a stream of notes of kdur duration on MIDI channel 1
  moscil 1, inote + k1, ivel,  kdur, kpause

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f0 60
</CsScore>
</CsoundSynthesizer>
```

## See Also

*midion, midion2, noteon, noteoff, noteondur, noteondur2*

## Credits

Author: Gabriel Maldonado  
Italy  
May 1997

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# mp3in

mp3in — Lit des données audio stéréo depuis un fichier MP3 externe.

## Description

Lit des données audio stéréo depuis un fichier MP3 externe.

## Syntaxe

```
ar1, ar2 mp3in ifilcod, iskptim, iformat, iskipinit, ibufsize
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son source. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) puis par SFDIR.

*iskptim* (facultatif) -- portion du son en entrée à ignorer, exprimée en secondes. La valeur par défaut est 0.

*iformat* (facultatif) -- spécifie le format des données du fichier audio : n'est pas encore implémenté et vaut stéréo par défaut.

*iskipinit* (facultatif) -- supprime toute initialisation s'il est non nul (vaut 0 par défaut).

*ibuffersize* (facultatif) -- fixe la taille du tampon de lecture interne. Si la valeur est zéro ou négative la taille par défaut est de 4096 octets.

## Exécution

Lit des données audio stéréo depuis un fichier MP3 externe.

## Voir Aussi

*diskin, ins, in, inh, inh, ino, inq, soundin*

## Crédits

Auteur : John ffitich  
Codemist Ltd  
2009

Nouveau dans la version 5.11

# mpulse

mpulse — Génère un ensemble d'impulsions.

## Description

Génère un ensemble d'impulsions d'amplitude *kamp* séparées par *kintvl* secondes (ou échantillons si *kintvl* est négatif). La première impulsion est générée après un délai de *ioffset* secondes.

## Syntaxe

```
ares mpulse kamp, kintvl [, ioffset]
```

## Initialisation

*ioffset* (facultatif, par défaut 0) -- le délai avant la première impulsion. S'il est négatif, la valeur est interprétée comme le nombre d'échantillons, sinon il représente des secondes. La valeur par défaut est zéro.

## Exécution

*kamp* -- amplitude des impulsions générées

*kintvl* -- intervalle de temps en secondes (ou en nombre d'échantillons si *kintvl* est négatif) jusqu'à la prochaine impulsion.

Après le délai initial, une impulsion d'amplitude *kamp* est générée comme échantillon unique. Immédiatement après la génération de l'impulsion, la date de la suivante est déterminée par la valeur de *kintvl* à ce moment précis. Cela signifie que tous les changements de *kintvl* entre les impulsions sont ignorés. Si *kintvl* est nul, il y a un temps d'attente infini jusqu'à la prochaine impulsion. Si *kintvl* est négatif, l'intervalle est compté en nombre d'échantillons plutôt qu'en secondes.

## Exemples

Voici un exemple de l'opcode mpulse. Il utilise le fichier *mpulse.csd* [examples/mpulse.csd].

### Exemple 319. Exemple de l'opcode mpulse.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o mpulse.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

gkfreq init 0.1

instr 1
  kamp = 10000

  a1 mpulse kamp, gkfreq
  out a1
endin

instr 2
; Assign the value of p4 to gkfreq
  gkfreq init p4
endin
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 10
i 2 2 1      0.05
i 2 4 1      0.01
i 2 6 1      0.005
i 2 8 1      0.001
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Ecrit par John ffitch.

Nouveau dans la version 4.08

Exemple écrit par Kevin Conder.

# mrtmsg

mrtmsg — Send system real-time messages to the MIDI OUT port.

## Description

Send system real-time messages to the MIDI OUT port.

## Syntax

```
mrtmsg msgtype
```

## Initialization

*msgtype* -- type of real-time message:

- 1 sends a START message (0xFA);
- 2 sends a CONTINUE message (0xFB);
- 0 sends a STOP message (0xFC);
- -1 sends a SYSTEM RESET message (0xFF);
- -2 sends an ACTIVE SENSING message (0xFE)

## Performance

Sends a real-time message once, in init stage of current instrument. *msgtype* parameter is a flag to indicate the message type.

## See Also

*mclock*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

# multitap

multitap — Ligne à retard avec plusieurs points de lecture.

## Description

Ligne à retard avec plusieurs points de lecture.

## Syntaxe

```
ares multitap asig [, itime1] [, igain1] [, itime2] [, igain2] [...]
```

## Initialisation

Les arguments *itime* et *igain* fixent la position et le gain de chaque point de lecture.

La ligne à retard est remplie par *asig*.

## Exemples

```
a1      oscil      1000, 100, 1
a2      multitap   a1, 1.2, .5, 1.4, .2
          out
a2
```

Cela produit deux délais, l'un de longueur 1.2 et de gain 0.5, et l'autre de longueur 1.4 et de gain 0.2

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1996



# mute

mute — Rend muettes/sonores de nouvelles instances d'un instrument donné.

## Description

Rend muettes/sonores de nouvelles instances d'un instrument donné.

## Syntaxe

```
mute insnum [, iswitch]
```

```
mute "insname" [, iswitch]
```

## Initialisation

*insnum* -- numéro d'instrument. Equivalent à *p1* dans une *instruction i* de partition.

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*iswitch* (facultatif, 0 par défaut) -- représente un commutateur pour rendre muet/sonore un instrument. Une valeur de 0 rendra muettes de nouvelles instances de l'instrument, tandis que les autres valeurs les rendront sonores. La valeur par défaut est 0.

## Exécution

Toutes les nouvelles instances de l'instrument seront muettes (*iswitch* = 0) ou sonores (*iswitch* différent de 0). Il n'y a aucun problème à rendre muets des instruments muets ou à rendre sonores des instruments sonores. Le mécanisme est le même que celui qui est utilisé par l'*instruction q*. de partition. Par exemple, il est possible de rendre muet depuis la partition et de rendre ensuite sonore depuis un instrument.

L'état Muet/Sonore est indiqué par un message (en fonction du niveau des messages).

## Exemples

Voici en exemple de l'opcode mute. Il utilise le fichier *mute.csd* [examples/mute.csd].

### Exemple 320. Exemple de l'opcode mute.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o mute.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
ksmps = 10
nchnls = 1
0dbfs = 1

; Mute Instrument #2.
mute 2
; Mute Instrument three.
mute "three"

; Instrument #1.
instr 1
  al oscils 0.2, 440, 0
  out al
endin

; Instrument #2.
instr 2
  al oscils 0.2, 880, 0
  out al
endin

; Instrument #3.
instr three
  al oscils 0.2, 1000, 0
  out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
; Play Instrument three for one second.
i three 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.22

## mxadsr

`mxadsr` — Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *expsegr*.

## Description

Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *expsegr*.

## Syntaxe

```
ares mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
```

```
kres mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
```

## Initialisation

*iatt* -- durée de l'attaque (attack)

*idec* -- durée de la première chute (decay)

*islev* -- niveau d'entretien (sustain)

*irel* -- durée de la chute (release)

*idel* (facultatif, 0 par défaut) -- délai de niveau zéro avant le démarrage de l'enveloppe

*ireltim* (facultatif, -1 par défaut) -- Contrôle la durée du relâchement après la réception d'un évènement MIDI note-off. S'il est inférieur à zéro, la durée de relâchement la plus longue de l'instrument courant est utilisée. S'il est nul ou positif, la valeur donnée sera utilisée comme durée de relâchement. Sa valeur par défaut est -1. (Nouveau dans Csound 3.59 - pas encore entièrement testé).

## Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

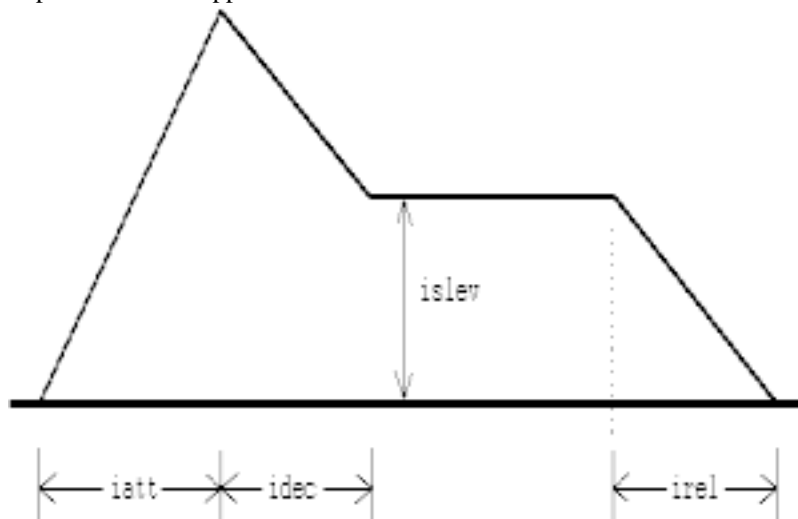


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des événements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et peut donc être utilisé dans les applications MIDI. L'opcode *mxadsr* est identique à *madsr* sauf qu'il utilise des segments exponentiels plutôt que linéaires.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *mxadsr*, car la durée est allongée automatiquement.

*mxadsr* est nouveau dans la version 3.51 de Csound.

## Voir Aussi

*linsegr*, *expsegr*, *envlpxr*, *mxadsr*, *madsr*, *adsr*, *expon*, *expseg*, *expsega* *line*, *linseg*, *xtratim*

## Crédits

Auteur : John ffitich

Novembre 2002. Merci à Rasmus Ekman pour avoir documenté le paramètre *ireltim*.

Novembre 2003. Merci à Kanata Motohashi pour avoir fixé le lien vers l'opcode *linsegr*.

# nchnls

nchnls — Fixe le nombre de canaux de la sortie audio.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

```
nchnls = iarg
```

## Initialisation

*nchnls* = (facultatif) -- fixe le nombre de canaux de la sortie audio à *iarg*. (1 = mono, 2 = stéréo, 4 = quadriphonique.) La valeur par défaut est 1 (mono).

De plus, toute *variable globale* [56] peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

## Voir Aussi

*kr*, *ksmps*, *sr*

# nestedap

nestedap — Three different nested all-pass filters.

## Description

Three different nested all-pass filters, useful for implementing reverbs.

## Syntax

```
ares nestedap asig, imode, imaxdel, idel1, igain1 [, idel2] [, igain2] \  
      [, idel3] [, igain3] [, istor]
```

## Initialization

*imode* -- operating mode of the filter:

- 1 = simple all-pass filter
- 2 = single nested all-pass filter
- 3 = double nested all-pass filter

*idel1*, *idel2*, *idel3* -- delay times of the filter stages. Delay times are in seconds and must be greater than zero. *idel1* must be greater than the sum of *idel2* and *idel3*.

*igain1*, *igain2*, *igain3* -- gain of the filter stages.

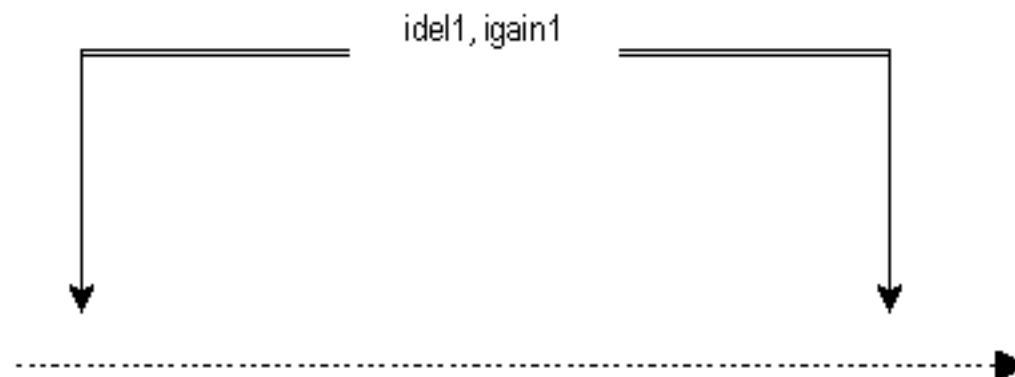
*imaxdel* -- will be necessary if k-rate delays are implemented. Not currently used.

*istor* -- Skip initialization if non-zero (default: 0).

## Performance

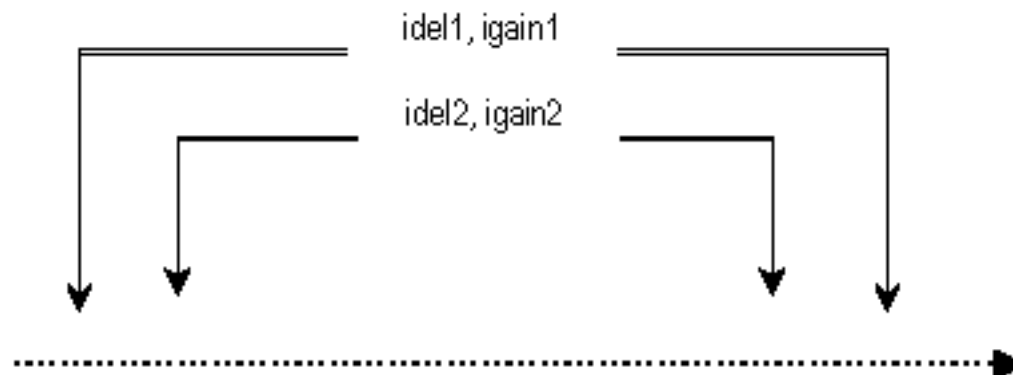
*asig* -- input signal

If *imode* = 1, the filter takes the form:



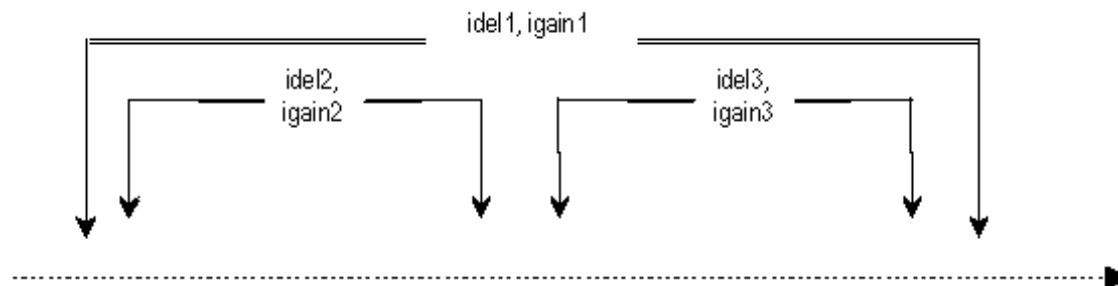
Picture of imode 1 filter.

If *imode* = 2, the filter takes the form:



Picture of imode 2 filter.

If *imode* = 3, the filter takes the form:



Picture of imode 3 filter.

## Examples

Here is an example of the nestedap opcode. It uses the file *nestedap.csd* [examples/nestedap.csd], and *beats.wav* [examples/beats.wav].

### Exemple 321. Example of the nestedap opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nestedap.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 5
```

```

    insnd      =      p4
    gasig      = diskin2 insnd, 1
endin

instr 10
    imax      =      1
    idel1     =      p4/1000
    igain1     =      p5
    idel2     =      p6/1000
    igain2     =      p7
    idel3     =      p8/1000
    igain3     =      p9
    idel4     =      p10/1000
    igain4     =      p11
    idel5     =      p12/1000
    igain5     =      p13
    idel6     =      p14/1000
    igain6     =      p15

    afdbk      = init 0

    aout1      = nestedap gasig+afdbk*.4, 3, imax, idel1, igain1, idel2, igain2, idel3, igain3
    aout2      = nestedap aout1, 2, imax, idel4, igain4, idel5, igain5
    aout       = nestedap aout2, 1, imax, idel6, igain6
    afdbk      = butterlp aout, 1000
               = outs gasig+(aout+aout1)/2, gasig-(aout+aout1)/2

gasig      =      0
endin

</CsInstruments>
<CsScore>

f1 0 8192 10 1

; Diskin
; Sta Dur Soundin
i5 0 3 "beats.wav"

; Reverb
; St Dur Del1 Gn1 Del2 Gn2 Del3 Gn3 Del4 Gn4 Del5 Gn5 Del6 Gn6
i10 0 4 97 .11 23 .07 43 .09 72 .2 53 .2 119 .3
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Hans Mikelson  
February 1999

New in Csound version 3.53

The example was updated May 2002, thanks to Hans Mikelson



# nlfilt

nlfilt — A filter with a non-linear effect.

## Description

Implements the filter:

$$Y\{n\} = a Y\{n-1\} + b Y\{n-2\} + d Y^2\{n-L\} + X\{n\} - C$$

described in Dobson and Fitch (ICMC'96)

## Syntax

ares **nlfilt** ain, ka, kb, kd, kC, kL

## Performance

1. Non-linear effect. The range of parameters are:

a = b = 0  
d = 0.8, 0.9, 0.7  
C = 0.4, 0.5, 0.6  
L = 20

This affects the lower register most but there are audible effects over the whole range. We suggest that it may be useful for coloring drums, and for adding arbitrary highlights to notes.

2. Low Pass with non-linear. The range of parameters are:

a = 0.4  
b = 0.2  
d = 0.7  
C = 0.11  
L = 20, ... 200

There are instability problems with this variant but the effect is more pronounced of the lower register, but is otherwise much like the pure comb. Short values of  $L$  can add attack to a sound.

3. High Pass with non-linear. The range of parameters are:

a = 0.35  
b = -0.3  
d = 0.95  
C = 0.2, ... 0.4

$L = 200$

4. High Pass with non-linear. The range of parameters are:

$a = 0.7$   
 $b = -0.2, \dots 0.5$   
 $d = 0.9$   
 $C = 0.12, \dots 0.24$   
 $L = 500, 10$

The high pass version is less likely to oscillate. It adds scintillation to medium-high registers. With a large delay  $L$  it is a little like a reverberation, while with small values there appear to be formant-like regions. There are arbitrary color changes and resonances as the pitch changes. Works well with individual notes.



### Warning

The "useful" ranges of parameters are not yet mapped.

## Credits

Author: John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
1997

New in version 3.44

# noise

noise — Un générateur de bruit blanc avec un filtre passe-bas à RII.

## Description

Un générateur de bruit blanc avec un filtre passe-bas à RII.

## Syntaxe

ares **noise** xamp, kbeta

## Exécution

*xamp* -- amplitude de la sortie finale

*kbeta* -- beta du filtre passe-bas. Doit être compris entre -1 et 1.

L'équation du filtre est :

$$y_n = \sqrt{(1 - \beta^2)} * x_n + \beta y_{(n-1)}$$

où  $x_n$  est le bruit blanc original et  $y_n$  est le bruit filtré. Plus  $\beta$  est élevé, plus basse est la fréquence de coupure du filtre. La fréquence de coupure vaut approximativement  $sr * ((1 - kbeta) / 2)$ .

## Exemples

Voici un exemple de l'opcode noise. Il utilise le fichier *noise.csd* [examples/noise.csd].

### Exemple 322. Exemple de l'opcode noise.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o noise.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
  kamp = 30000

  ; Change the beta value linearly from 0 to 1.
  kbeta line 0, p3, 1

  al noise kamp, kbeta
  out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode noise dans lequel on contrôle le paramètre *kbeta* au moyen d'une interface graphique. Il utilise le fichier *noise-2.csd* [examples/noise-2.csd].

### Exemple 323. Exemple de l'opcode noise contrôlé par une interface graphique.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in No messages
-odac ; -iadc -d ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o noise.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

FLpanel "noise", 200, 50, -1, -1
  gkbeta, gslider1 FLslider "kbeta", -1, 1, 0, 5, -1, 180, 20, 10, 10
FLpanelEnd
FLrun

instr 1
  iamp = 0dbfs / 4 ; Peaks 12 dB below 0dbfs
  print iamp

  al noise iamp, gkbeta
  printk2 gkbeta
  outs al,al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute.
i 1 0 60
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Décembre 2000

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.10 de Csound

# noteoff

noteoff — Send a noteoff message to the MIDI OUT port.

## Description

Send a noteoff message to the MIDI OUT port.

## Syntax

```
noteoff ichn, inum, ivel
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*inum* -- note number (0-127)

*ivel* -- velocity (0-127)

## Performance

*noteon* (i-rate note on) and *noteoff* (i-rate note off) are the simplest MIDI OUT opcodes. *noteon* sends a MIDI noteon message to MIDI OUT port, and *noteoff* sends a noteoff message. A *noteon* opcode must always be followed by an *noteoff* with the same channel and number inside the same instrument, otherwise the note will play endlessly.

These *noteon* and *noteoff* opcodes are useful only when introducing a *timeout* statement to play a non-zero duration MIDI note. For most purposes, it is better to use *noteondur* and *noteondur2*.

## See Also

*noteon*, *noteondur*, *noteondur2*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# noteon

noteon — Send a noteon message to the MIDI OUT port.

## Description

Send a noteon message to the MIDI OUT port.

## Syntax

```
noteon ichn, inum, ivel
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*inum* -- note number (0-127)

*ivel* -- velocity (0-127)

## Performance

*noteon* (i-rate note on) and *noteoff* (i-rate note off) are the simplest MIDI OUT opcodes. *noteon* sends a MIDI noteon message to MIDI OUT port, and *noteoff* sends a noteoff message. A *noteon* opcode must always be followed by an *noteoff* with the same channel and number inside the same instrument, otherwise the note will play endlessly.

These *noteon* and *noteoff* opcodes are useful only when introducing a *timeout* statement to play a non-zero duration MIDI note. For most purposes, it is better to use *noteondur* and *noteondur2*.

## See Also

*noteoff*, *noteondur*, *noteondur2*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# noteondur

*noteondur* — Sends a noteon and a noteoff MIDI message both with the same channel, number and velocity.

## Description

Sends a noteon and a noteoff MIDI message both with the same channel, number and velocity.

## Syntax

```
noteondur ichn, inum, ivel, idur
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*inum* -- note number (0-127)

*ivel* -- velocity (0-127)

*idur* -- how long, in seconds, this note should last.

## Performance

*noteondur* (i-rate note on with duration) sends a noteon and a noteoff MIDI message both with the same channel, number and velocity. Noteoff message is sent after *idur* seconds are elapsed by the time *noteondur* was active.

*noteondur* differs from *noteondur2* in that *noteondur* truncates note duration when current instrument is deactivated by score or by real-time playing, while *noteondur2* will extend performance time of current instrument until *idur* seconds have elapsed. In real-time playing, it is suggested to use *noteondur* also for undefined durations, giving a large *idur* value.

Any number of *noteondur* opcodes can appear in the same Csound instrument, allowing chords to be played by a single instrument.

## Examples

Here is an example of the *noteondur* opcode. It uses the file *noteondur.csd* [examples/noteondur.csd].

### Exemple 324. Example of the *noteondur* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates notes for every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```
<CsoundSynthesizer>  
<CsOptions>
```



```

; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d          -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ;Turned on by MIDI notes on channel 1

    ifund    notnum
    ivel      veloc
    idur = 1

    ;chord with single key
    noteondur    1, ifund,    ivel, idur
    noteondur    1, ifund+3, ivel, idur
    noteondur    1, ifund+7, ivel, idur
    noteondur    1, ifund+9, ivel, idur

endin

</CsInstruments>
<CsScore>
; Play Instrument #1 for 60 seconds.

i1 0 60

</CsScore>
</CsoundSynthesizer>

```

## See Also

*noteoff, noteon, noteondur2, midion, midion2*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# noteondur2

**noteondur2** — Sends a noteon and a noteoff MIDI message both with the same channel, number and velocity.

## Description

Sends a noteon and a noteoff MIDI message both with the same channel, number and velocity.

## Syntax

```
noteondur2 ichn, inum, ivel, idur
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*inum* -- note number (0-127)

*ivel* -- velocity (0-127)

*idur* -- how long, in seconds, this note should last.

## Performance

*noteondur2* (i-rate note on with duration) sends a noteon and a noteoff MIDI message both with the same channel, number and velocity. Noteoff message is sent after *idur* seconds are elapsed by the time *noteondur2* was active.

*noteondur* differs from *noteondur2* in that *noteondur* truncates note duration when current instrument is deactivated by score or by real-time playing, while *noteondur2* will extend performance time of current instrument until *idur* seconds have elapsed. In real-time playing, it is suggested to use *noteondur* also for undefined durations, giving a large *idur* value.

Any number of *noteondur2* opcodes can appear in the same Csound instrument, allowing chords to be played by a single instrument.

## Examples

Here is an example of the *noteondur2* opcode. It uses the file *noteondur2.csd* [examples/noteondur2.csd].

### Exemple 325. Example of the *noteondur2* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates notes for every note received on the MIDI input. It generates MIDI notes on csound's MIDI output, so be sure to connect something.

```

<CsoundSynthesizer>
<CsOptions>

; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d          -M0  -Q1;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1

    ifund    notnum
    ivel      veloc
    idur = 1

    ;chord with single key
    noteondur2 1, ifund,   ivel, idur
    noteondur2 1, ifund+3, ivel, idur
    noteondur2 1, ifund+7, ivel, idur
    noteondur2 1, ifund+9, ivel, idur

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>

```

## See Also

*noteoff, noteon, noteondur, midion, midion2*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# notnum

notnum — Donne un numéro de note à partir d'un évènement MIDI.

## Description

Donne un numéro de note à partir d'un évènement MIDI.

## Syntaxe

ival **notnum**

## Exécution

Donne la valeur de l'octet MIDI (0 - 127) représentant le numéro de note de l'évènement courant.

## Exemples

Voici un exemple de l'opcode notnum. Il utilise le fichier *notnum.csd* [examples/notnum.csd].

### Exemple 326. Exemple de l'opcode notnum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o notnum.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  il notnum

  print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e

</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode `notnum` utilisé pour produire une sortie audio sortie. Il utilise le fichier `notnum_complex.csd` [examples/notnum\_complex.csd]

### Exemple 327. Exemple complexe de l'opcode `notnum`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr      =      44100
ksmps   =      10
nchnls  =      2

; Set MIDI channel 1 to play instr 1.
      massign 1, 1

      instr      1

; Returns MIDI note number - an integer in range (0-127)
iNum      notnum

; Convert MIDI note number to Hz
iHz      = (440.0*exp(log(2.0)*((iNum)-69.0)/12.0))

; Generate audio by indexing a table; fixed amplitude.
aosc      oscil      10000, iHz, 1

; Since there is no enveloping, there will be clicks.
outs      aosc, aosc

      endin

</CsInstruments>
<CsScore>

; Generate a Sine-wave to be indexed at audio rate
; by the oscil opcode.
f1      0      16384      10      1

; Keep the score "open" for 1 hour so that MIDI
; notes can allocate new note events, arbitrarily.
f0      3600

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemples écrits par Kevin Conder et David Akbari.

# nreverb

nreverb — A reverberator consisting of 6 parallel comb-lowpass filters.

## Description

This is a reverberator consisting of 6 parallel comb-lowpass filters being fed into a series of 5 allpass filters. *nreverb* replaces *reverb2* (version 3.48) and so both opcodes are identical.

## Syntax

```
ares nreverb asig, ktime, khdif [, iskip] [,inumCombs] [, ifnCombs] \  
      [, inumAlpas] [, ifnAlpas]
```

## Initialization

*iskip* (optional, default=0) -- Skip initialization if present and non-zero.

*inumCombs* (optional) -- number of filter constants in comb filter. If omitted, the values default to the nreverb constants. New in Csound version 4.09.

*ifnCombs* - function table with *inumCombs* comb filter time values, followed the same number of gain values. The ftable should not be rescaled (use negative fgen number). Positive time values are in seconds. The time values are converted internally into number of samples, then set to the next greater prime number. If the time is negative, it is interpreted directly as time in sample frames, and no processing is done (except negation). New in Csound version 4.09.

*inumAlpas*, *ifnAlpas* (optional) -- same as *inumCombs/ifnCombs*, for allpass filter. New in Csound 4.09.

## Performance

The input signal *asig* is reverberated for *ktime* seconds. The parameter *khdif* controls the high frequency diffusion amount. The values of *khdif* should be from 0 to 1. If *khdif* is set to 0 the all the frequencies decay with the same speed. If *khdif* is 1, high frequencies decay faster than lower ones. If *ktime* is inadvertently set to a non-positive number, *ktime* will be reset automatically to 0.01. (New in Csound version 4.07.)

As of Csound version 4.09, *nreverb* may read any number of comb and allpass filter from an ftable.

## Examples

Here is a simple example of the nreverb opcode. It uses the file *nreverb.csd* [examples/nreverb.csd].

### Exemple 328. Simple example of the nreverb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nreverb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
  a1 oscil 10000, 440, 1
  a2 nreverb a1, 2.5, .3
  out a1 + a2 * .2
endin

</CsInstruments>
<CsScore>

; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

i 1 0.0 0.5
i 1 1.0 0.5
i 1 2.0 0.5
i 1 3.0 0.5
i 1 4.0 0.5
e

</CsScore>
</CsoundSynthesizer>

```

Here is an example of the `nreverb` opcode using an `ftable` for filter constants. It uses the file `nreverb_ftable.csd` [examples/nreverb\_ftable.csd], and `beats.wav` [examples/beats.wav].

### Exemple 329. An example of the `nreverb` opcode using an `ftable` for filter constants.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nreverb_ftable.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
  a1 soundin "beats.wav"
  a2 nreverb a1, 1.5, .75, 0, 8, 71, 4, 72
  out a1 + a2 * .4
endin

</CsInstruments>
<CsScore>

; freeverb time constants, as direct (negative) sample, with arbitrary gains
f71 0 16 -2 -1116 -1188 -1277 -1356 -1422 -1491 -1557 -1617 0.8 0.79 0.78 0.77 0.76 0.75 0.74

f72 0 16 -2 -556 -441 -341 -225 0.7 0.72 0.74 0.76

```

```
i1 0 3  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Credits

Authors: Paris Smaragdis (*reverb2*)  
MIT, Cambridge  
1995

Author: Richard Karpen (*nreverb*)  
Seattle, Wash  
1998



## nrpn

**nrpn** — Sends a Non-Registered Parameter Number to the MIDI OUT port.

## Description

Sends a NPRN (Non-Registered Parameter Number) message to the MIDI OUT port each time one of the input arguments changes.

## Syntax

**nrpn** *kchan*, *kparmnum*, *kparmvalue*

## Performance

*kchan* -- MIDI channel (1-16)

*kparmnum* -- number of NRPN parameter

*kparmvalue* -- value of NRPN parameter

This opcode sends new message when the MIDI translated value of one of the input arguments changes. It operates at k-rate. Useful with the MIDI instruments that recognize NRPNs (for example with the newest sound-cards with internal MIDI synthesizer such as SB AWE32, AWE64, GUS etc. in which each patch parameter can be changed during the performance via NRPN)

## Credits

Author: Gabriel Maldonado  
Italy  
1998

New in Csound version 3.492

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# nsamp

nsamp — Retourne le nombre d'échantillons chargés dans une table de fonction.

## Description

Retourne le nombre d'échantillons chargés dans une table de fonction.

## Syntaxe

**nsamp**(x) (arg de taux-i seulement)

## Exécution

Retourne le nombre d'échantillons chargés dans la table de fonction numéro *x* par *GEN01*. Utile lorsqu'un échantillon est plus court que la puissance de deux, taille de la table de fonction qui le contient. Nouveau dans la version 3.49 de Csound.

A partir de la version 5.02 de Csound, *nsamp* travaille avec les tables de fonction à longueur différée (voir *GEN01*).

*nsamp* diffère de *flen* en ce sens que *nsamp* donne le nombre de trames d'échantillon chargées, tandis que *flen* donne le nombre total d'échantillons. Par exemple, avec un fichier son stéréo de 10000 échantillons, *flen*() retournera 19999 (c'est-à-dire un total de 20000 échantillons mono, en excluant le point de garde), mais *nsamp*() retournera 10000.

## Exemples

Voici un exemple de l'opcode *nsamp*. Il utilise les fichiers *nsamp.csd* [examples/nsamp.csd] et *mary.wav* [examples/mary.wav].

### Exemple 330. Exemple de l'opcode nsamp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nsamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the size (in samples) of Table #1.
isz = nsamp(1)
```

```
    print isz
  endin

</CsInstruments>
<CsScore>

; Table #1: Use an audio file.
f 1 0 262144 1 "mary.wav" 0 0 0

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Comme le fichier son « mary.wav » a 154390 échantillons, la sortie comprendra une ligne comme celle-ci :

```
instr 1:  isz = 154390.000
```

## Voir Aussi

*ftchnls, flen, flptim, ftr*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Octobre 1998

Exemple écrit par Kevin Conder.

# nstrnum

nstrnum — Retourne le numéro d'un instrument nommé.

## Description

Retourne le numéro d'un instrument nommé.

## Syntaxe

```
insno nstrnum "name"
```

## Initialisation

*insno* -- le numéro de l'instrument nommé.

## Exécution

*"name"* -- le nom de l'instrument nommé.

Si aucun instrument n'existe avec le nom spécifié, une erreur d'initialisation survient, et la valeur -1 est retournée.

## Crédits

Auteur : Istvan Varga  
Nouveau dans la version 4.23  
Ecrit en 2002.

# ntrpol

ntrpol — Calculates the weighted mean value of two input signals.

## Description

Calculates the weighted mean value (i.e. linear interpolation) of two input signals

## Syntax

```
ares ntrpol asig1, asig2, kpoint [, imin] [, imax]
```

```
ires ntrpol isig1, isig2, ipoint [, imin] [, imax]
```

```
kres ntrpol ksig1, ksig2, kpoint [, imin] [, imax]
```

## Initialization

*imin* -- minimum xpoint value (optional, default 0)

*imax* -- maximum xpoint value (optional, default 1)

## Performance

*xsig1*, *xsig2* -- input signals

*xpoint* -- interpolation point between the two values

*ntrpol* opcode outputs the linear interpolation between two input values. *xpoint* is the distance of evaluation point from the first value. With the default values of *imin* and *imax*, (0 and 1) a zero value indicates no distance from the first value and the maximum distance from the second one. With a 0.5 value, *ntrpol* will output the mean value of the two inputs, indicating the exact half point between *xsig1* and *xsig2*. A 1 value indicates the maximum distance from the first value and no distance from the second one. The range of *xpoint* can be also defined with *imin* and *imax* to make its management easier.

These opcodes are useful for crossfading two signals.

## Credits

Author: Gabriel Maldonado  
Italy  
October 1998

New in Csound version 3.49

# octave

octave — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre d'octaves.

## Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre d'octaves.

## Syntaxe

`octave(x)`

Cette fonction travaille aux taux-i, -k et -a.

## Initialisation

*x* -- une valeur exprimée en octaves.

## Exécution

La valeur retournée par la fonction *octave* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre d'octaves spécifié.

## Exemples

Voici un exemple de l'opcode octave. Il utilise le fichier *octave.csd* [examples/octave.csd].

### Exemple 331. Exemple de l'opcode octave.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o octave.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; The root note is A above middle-C (440 Hz)
iroot = 440

; Raise the root note by two octaves.
ioctaves = 2

; Calculate the new note.
```

```
ifactor = octave(ioctaves)
inew = iroot * ifactor

; Print out of all of the values.
print iroot
print ifactor
print inew
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1: iroot = 440.000
instr 1: ifactor = 4.000
instr 1: inew = 1760.149
```

## Voir Aussi

*cent, db, semitone*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.16

# octcps

octcps — Convertit des cycles par seconde en valeur octave-point-partie-décimale.

## Description

Convertit des cycles par seconde en valeur octave-point-partie-décimale.

## Syntaxe

`octcps (cps) (arguments de taux-i ou -k seulement)`

où l'argument entre parenthèses peut être une expression.

## Exécution

*octcps* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 17. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.





## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *octcps*. Il utilise le fichier *octcps.csd* [examples/octcps.csd].

### Exemple 332. Exemple de l'opcode *octcps*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o octcps.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Convert a cycles-per-second value into an
; octave value.
icps = 440
ioct = octcps(icps)

print ioct
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  ioct = 8.750
```

## Voir Aussi

*cpsoct, cpspch, octpch, pchoct, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Exemple écrit par Kevin Conder.

# octmidi

octmidi — Get the note number, in octave-point-decimal units, of the current MIDI event.

## Description

Get the note number, in octave-point-decimal units, of the current MIDI event.

## Syntax

ioct **octmidi**

## Performance

Get the note number of the current MIDI event, expressed in octave-point-decimal units, for local processing.



### octmidi vs. octmidinn

The *octmidi* opcode only produces meaningful results in a Midi-activated note (either real-time or from a Midi score with the -F flag). With *octmidi*, the Midi note number value is taken from the Midi event that is internally associated with the instrument instance. On the other hand, the *octmidinn* opcode may be used in any Csound instrument instance whether it is activated from a Midi event, score event, line event, or from another instrument. The input value for *octmidinn* might for example come from a p-field in a textual score or it may have been retrieved from the real-time Midi event that activated the current note using the *notnum* opcode.

## Examples

Here is an example of the octmidi opcode. It uses the file *octmidi.csd* [examples/octmidi.csd].

### Exemple 333. Example of the octmidi opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac        -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o octmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
; This example expects MIDI note inputs on channel 1
il octmidi

print i1
endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidib, pchbend, pchmidi, pchmidib, veloc, cpsmidinn, octmidinn, pchmidinn*

## Credits

Author: Barry L. Vercoe - Mike Berry  
MIT - Mills  
May 1997

Example written by Kevin Conder.

# octmidib

octmidib — Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in octave-point-decimal.

## Description

Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in octave-point-decimal.

## Syntax

```
ioct octmidib [irange]
```

```
koct octmidib [irange]
```

## Initialization

*irange* (optional) -- the pitch bend range in semitones

## Performance

Get the note number of the current MIDI event, modify it by the current pitch-bend value, and express the result in octave-point-decimal units. Available as an i-time value or as a continuous k-rate value.

## Examples

Here is an example of the octmidib opcode. It uses the file *octmidib.csd* [examples/octmidib.csd].

### Exemple 334. Example of the octmidib opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o octmidib.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; This example expects MIDI note inputs on channel 1
i1 octmidib
```

```
    print i1
  endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, pchbend, pchmidi, pchmidib, veloc*

## Credits

Author: Barry L. Vercoe - Mike Berry  
MIT - Mills  
May 1997

Example written by Kevin Conder.

# octmidinn

octmidinn — Convertit un numéro de note Midi en octave-point-partie-décimale.

## Description

Convertit un numéro de note Midi en octave-point-partie-décimale.

## Syntaxe

**octmidinn** (MidiNoteNumber) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*octmidinn* est une fonction qui prend une valeur de taux-i ou de taux-k représentant un numéro de note Midi et qui retourne la valeur de hauteur équivalente dans le format octave-point-partie-décimale de Csound. Cette conversion suppose que le do médian (8.000 en *oct*) est la note Midi numéro 60. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.



### octmidinn vs. octmidi

L'opcode *octmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un événement Midi, un événement de partition, un événement en ligne, ou depuis un autre instrument. La valeur d'entrée de *octmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode *notnum* à partir de l'évènement Midi en temps-réel qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de taux-i ou de taux-k. D'un autre côté, l'opcode *octmidi* ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option -F). Avec *octmidi*, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

*octmidinn* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 18. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *octmidinn*. Il utilise le fichier *cpsmidinn.csd* [exemples/cpsmidinn.csd].

### Exemple 335. Exemple de l'opcode *octmidinn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
; i-time loop to print conversion table
imidiNN = 0
loop1:
  icps = cpsmidinn(imidiNN)
  ioct = octmidinn(imidiNN)
  ipch = pchmidinn(imidiNN)

  print imidiNN, icps, ioct, ipch

  imidiNN = imidiNN + 1
  if (imidiNN < 128) igoto loop1
```



```

endin

instr 2
; test k-rate converters
kMiddleC = 60
kcps = cpsmidinn(kMiddleC)
koct = octmidinn(kMiddleC)
kpch = pchmidinn(kMiddleC)

printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*cpsmidinn, pchmidinn, octmidi, notnum, cpspch, cpsoct, octcps, octpch, pchoct*

## Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07

# octpch

octpch — Convertit une valeur de classe de hauteur en octave-point-partie-décimale.

## Description

Convertit une valeur de classe de hauteur en octave-point-partie-décimale.

## Syntaxe

`octpch` (`pch`) (`arguments de taux-i ou -k seulement`)

où l'argument entre parenthèses peut être une expression.

## Exécution

*octpch* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 19. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + *k1*) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque *k*-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *octpch*. Il utilise le fichier *octpch.csd* [examples/octpch.csd].

### Exemple 336. Exemple de l'opcode *octpch*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o octpch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Convert a pitch-class value into an
; octave-point-decimal value.
ipch = 8.09
ioct = octpch(ipch)

print ioct
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  ioct = 8.750
```

## Voir Aussi

*cpsoct, cpspch, octcps, pchoct, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Exemple écrit par Kevin Conder.

# opcode

opcode — Commence un bloc d'opcode défini par l'utilisateur.

## Définir des opcodes

Les instructions *opcode* et *endop* permettent de définir un nouvel opcode qui peut être utilisé de la même façon qu'un opcode original de Csound. Ces blocs d'opcode ressemblent beaucoup aux instruments (et sont, en fait, implémentés comme des instruments spéciaux), mais on ne peut pas les appeler comme des instruments normaux, par exemple avec des *instructions i*.

Un bloc d'opcode défini par l'utilisateur doit précéder l'instrument (ou l'opcode) depuis lequel on l'utilise. Mais un opcode peut aussi s'appeler lui-même. Cela permet une récursivité dont la profondeur n'est limitée que par la mémoire disponible. De plus, on peut, à titre expérimental, exécuter l'opcode défini à un taux de contrôle plus élevé que la valeur de *kr* spécifiée dans l'en-tête de l'orchestre.

Comme pour les instruments, les variables et les étiquettes d'un bloc d'opcode défini par l'utilisateur sont locales et ne sont pas visible depuis l'instrument appelant (de même que l'opcode n'a pas accès aux variables de l'instrument qui l'a appelé).

Cependant, certains paramètres sont copiés automatiquement à l'initialisation :

- tous les p-champs (*p1* inclus)
- le temps supplémentaire (voir aussi *xtratim*, *linsegr*, et les opcodes correspondants). Ceci peut affecter le fonctionnement de *linsegr/expsegr/linenr/envlpxr* dans le bloc d'opcode défini par l'utilisateur.
- les paramètres MIDI, s'il y en a.

Le drapeau de release (voir l'opcode *release*) est également copié durant l'exécution.

La modification de la durée de la note dans la définition de l'opcode en assignant une valeur à *p3*, ou l'utilisation de *ihold*, *turnoff*, *xtratim*, *linsegr*, ou d'autres opcodes similaires affecteront aussi l'instrument appelant. Les changements sur des contrôleurs MIDI (par exemple avec *ctrlinit*) s'appliqueront aussi à l'instrument qui a appelé l'opcode.

Utilisez l'opcode *setksmps* pour fixer la valeur locale de *ksmps*.

Les opcodes *xin* et *xout* copient des variables vers et depuis la définition de l'opcode, permettant la communication avec l'instrument appelant.

Les types des variables d'entrée et de sortie sont définis par les paramètres *intypes* et *outtypes*.



## Notes

- *xin* et *xout* ne doivent être appelés qu'une seule fois, et *xin* doit précéder *xout*, sinon une erreur d'initialisation et une désactivation de l'instrument courant peuvent se produire.
- Ces deux opcodes n'agissent qu'à l'initialisation. La copie durant l'exécution est réalisée par l'appel de l'opcode de l'utilisateur. Cela signifie que sauter *xin* ou *xout* avec *kgoto* n'a aucun effet, alors que les sauter avec *igoto* affecte à la fois les opérations de l'initialisation et de l'exécution.

## Syntaxe

`opcode nom, outtypes, intypes`

## Initialisation

*nom* -- nom de l'opcode. Il est constitué de n'importe quelle combinaison de lettres, chiffres et traits de soulignement mais il ne doit pas commencer par un chiffre. Si un opcode du même nom existe déjà, il est redéfini (un avertissement est imprimé dans ce cas). Certains mots réservés (comme *instr* et *endin*) ne peuvent pas être redéfinis.

*intypes* -- liste des types en entrée, combinaison de caractères pris parmi : a, k, K, i, o, p, et j. Un caractère 0 unique peut être utilisé s'il n'y a pas d'argument en entrée. Il n'y a *pas* besoin d'apostrophes doubles et de délimiteurs (comme la virgule).

La signification des différent *intypes* est montrée dans le tableau suivant :

Type	Description	Types de Variable Autorisés	Mise à jour
a	variable de taux-a	taux-a	taux-a
i	variable de taux-i	taux-i	initialisation
j	facultatif de taux-i, -1 par défaut	taux-i, constante	initialisation
k	variable de taux-k	taux-k et -i, constante	taux-k
K	taux-k avec initialisation	taux-k et -i, constante	taux-i et taux-k
o	facultatif à l'initialisation, 0 par défaut	taux-i, constante	initialisation
p	facultatif à l'initialisation, 1 par défaut	taux-i, constante	initialisation
S	variable chaîne de caractères	chaîne de caractères de taux-i	initialisation

Le nombre maximum d'arguments en entrée autorisé est 256.

*outtypes* -- liste des types en sortie. Le format est le même que celui utilisé pour *intypes*.

Voici les *outtypes* disponibles :

Type	Description	Types de Variable Autorisés	Mise à jour
a	variable de taux-a	taux-a	taux-a
i	variable de taux-i	taux-i	initialisation
k	variable de taux-k	taux-k	taux-k
K	taux-k avec initialisation	taux-k	taux-i et taux-k

Le nombre maximum d'arguments en sortie autorisé est 256.

*iksmips* (facultatif, 0 par défaut) -- fixe la valeur locale de *ksmps*. Doit être un nombre entier positif, et le *ksmps* de l'instrument appelant doit être un multiple entier de cette valeur. Par exemple, si *ksmps* vaut 10 dans l'instrument depuis lequel l'opcode a été appelé, les valeurs permises pour *iksmips* sont 1, 2, 5, et 10.

Si *iksmips* vaut zéro, le *ksmps* de l'instrument ou de l'opcode appelant est utilisé (c'est le comportement par défaut).



## Note

Le *ksmps* local est implémenté en divisant une période de contrôle en sous-périodes-k plus petites et en modifiant temporairement les variables globales internes de Csound. Ceci nécessite aussi la conversion du taux des arguments d'entrée et de sortie de taux-k (les variables d'entrée reçoivent la même valeur dans tous les sous-périodes-k, tandis que les valeurs de sortie ne sont écrites que pendant la dernière).



## Avertissement au sujet du *ksmps* local

Lorsque le *ksmps* local est différent du *ksmps* de l'orchestre (celui spécifié dans l'en-tête de l'orchestre), il ne faut pas utiliser d'opération globale de taux-a dans le bloc d'opcode défini par l'utilisateur.

Ceci comprend :

- tous les accès aux variables « ga »
- les opcodes zak de taux-a (*zar*, *zaw*, etc.)
- *tablera* et *tablewa* (ces deux opcodes peuvent fonctionner en fait, mais il faut prendre des précautions)
- La famille d'opcode *in* et *out* (ils lisent depuis et écrivent dans des tampons globaux de taux-a)

En général, il faut utiliser le *ksmps* local avec précaution car c'est une fonctionnalité expérimentale, bien qu'elle fonctionne correctement dans la plupart des cas.

L'instruction *setksmps* peut être utilisée pour fixer la valeur du *ksmps* local du bloc d'opcode défini par l'utilisateur. Elle a un paramètre de taux-i spécifiant la nouvelle valeur de *ksmps* (qui reste inchangée si l'on utilise zéro, voir aussi les notes au sujet de *iksmips* ci-dessus). *setksmps* doit être utilisé avant tout autre opcode (mais il est autorisé après *xin*), autrement des résultats imprévisibles peuvent se produire.

On peut lire les paramètres d'entrée avec l'opcode *xin*, et la sortie est écrite par l'opcode *xout*. On ne doit utiliser qu'une seule instance de ces unités, car *xout* écrase la sortie sans accumuler les valeurs. Le nombre et le type des arguments pour *xin* et *xout* doit être le même que dans la déclaration du bloc d'opcode défini par l'utilisateur (voir les tableaux ci-dessus).

Les arguments d'entrée et de sortie doivent se conformer à la définition à la fois en nombre (sauf si des entrées de taux-i facultatives sont utilisées) et en genre. Un paramètre d'entrée facultatif de taux-i (*iksmips*) est automatiquement ajouté à la liste des *intypes* et (comme pour *setksmps*) fixe la valeur du *ksmps* local.

## Exécution

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode nom, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

Le nouvel opcode peut ensuite être utilisé avec la syntaxe usuelle :

```
[xoutarg1] [, xoutarg2] ... [xoutargN] nom [xinarg1] [, xinarg2] ... [xinargN] [, iksmps]
```



## Note

L'opcode est toujours appelé à la fois durant l'initialisation et durant l'exécution, même s'il n'y a pas d'arguments de taux-k ou -a. Si l'on sait que plusieurs opcodes définis par l'utilisateur n'ont pas d'effet durant l'exécution (taux-k) dans un instrument, on peut épargner du temps CPU en sautant ces groupes d'opcodes avec *kgoto*.

## Exemples

Voici un exemple d'opcode défini par l'utilisateur. Il utilise le fichier *opcode.csd* [examples/op-code\_example.csd].

### Exemple 337. Exemple d'opcode défini par l'utilisateur.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac -iadc ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o opcode_example.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 50
nchnls  = 1

/* example opcode 1: simple oscillator */

opcode Oscillator, a, kk

kamp, kcps xin          ; read input parameters
al      vco2 kamp, kcps  ; sawtooth oscillator
xout al  ; write output

endop

/* example opcode 2: lowpass filter with local ksmps */

opcode Lowpass, a, akk

      setksmps 1          ; need sr=kr
ain, kal, ka2 xin         ; read input parameters
aout    init 0           ; initialize output
aout    = ain*kal + aout*ka2 ; simple tone-like filter
xout aout ; write output
```



```

        endop

/* example opcode 3: recursive call */

        opcode RecursiveLowpass, a, akkpp

ain, kal, ka2, idep, icnt      xin      ; read input parameters
        if (icnt >= idep) goto skip1    ; check if max depth reached
ain      RecursiveLowpass ain, kal, ka2, idep, icnt + 1
skip1:
aout     Lowpass ain, kal, ka2      ; call filter
        xout aout                  ; write output

        endop

/* example opcode 4: de-click envelope */

        opcode DeClick, a, a

ain      xin
aenv     linseg 0, 0.02, 1, p3 - 0.05, 1, 0.02, 0, 0.01, 0
        xout ain * aenv            ; apply envelope and write output

        endop

/* instr 1 uses the example opcodes */

instr 1

kamp     = 20000                    ; amplitude
kcps     expon 50, p3, 500          ; pitch
al       Oscillator kamp, kcps      ; call oscillator
kflt     linseg 0.4, 1.5, 0.4, 1, 0.8, 1.5, 0.8 ; filter envelope
al       RecursiveLowpass al, kflt, 1 - kflt, 10 ; 10th order lowpass
al       DeClick al
        out al

        endin

</CsInstruments>
<CsScore>

i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*endop, setksmps, xin, xout*

## Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code de Matt J. Ingalls

Nouveau dans la version 4.22

# OSCsend

OSCsend — Envoie des données à d'autres processus au moyen du protocole OSC.

## Description

Utilise le protocole OSC pour envoyer un message à d'autres processus d'écoute OSC.

## Syntaxe

```
OSCsend kwhen, ihost, iport, idestination, itype [, kdata1, kdata2, ...]
```

## Initialisation

*ihost* -- une chaîne de caractères donnant le nom de domaine de l'ordinateur hôte destinataire. Une chaîne vide est interprétée comme l'ordinateur courant.

*iport* -- le numéro du port utilisé pour la communication.

*idest* -- une chaîne de caractères indiquant l'adresse de destination. Elle prend la forme d'un nom de fichier avec des répertoires. Csound ne fait que transmettre cette chaîne au code brut envoyé sans faire d'interprétation.

*itype* -- une chaîne de caractères indiquant le type des arguments facultatifs qui sont lus au taux-k. La chaîne peut contenir les caractères "bcdfilmst" pour booléen, caractère, double, flottant, entier sur 32 bit, entier sur 64 bit, MIDI, chaîne de caractères et repère temporelle.

## Exécution

*kwhen* -- un message est envoyé chaque fois que cette valeur change. Un message sera toujours envoyé au premier appel.

Les données proviennent des valeurs de taux-k qui suivent la chaîne de formatage. De même que pour le format dans printf, la série de caractères détermine l'interprétation des arguments. Noter qu'un repère temporel prend deux arguments.

## Exemple

L'exemple montre un simple instrument qui, lorsqu'il est appelé, envoie un groupe de trois messages à un ordinateur nommé "xenakis", sur le port 7770, à lire par un processus dont l'adresse est /foo/bar.

```
instr 1
  OSCsend 1, "xenakis.cs.bath.ac.uk", 7770, "/foo/bar", "sis", "FOO", 42, "bar"
endin
```

Voir la notice d'*OSClisten* pour un exemple d'envoi/réception en utilisant OSC.

## Voir Aussi

*OSListen, OSCinit*

## Crédits

Auteur : John ffitch  
2005

# OSCinit

OSCinit — Démarre l'écoute des messages OSC sur un port particulier.

## Description

Démarre un processus d'écoute qui peut être utilisé par *OSClisten*.

## Syntaxe

```
ihandle OSCinit iport
```

## Initialisation

*ihandle* -- identifiant retourné que l'on peut passer à n'importe quel nombre d'opcodes *OSClisten* pour recevoir des messages sur ce port.

*iport* -- le port sur lequel on écoute.

## Exécution

Le module d'écoute fonctionne en tâche de fond. Voir *OSClisten* pour les détails.

## Exemple

Cet exemple montre une paire de nombres en virgule flottante reçus sur le port 7770.

```
sr = 44100
ksmps = 100
nchnls = 2

gihandle OSCinit 7770

    instr    1
    kf1 init 0
    kf2 init 0
nxtmsg:
    kk OSClisten gihandle, "/foo/bar", "ff", kf1, kf2
if (kk == 0) goto ex
    printk 0,kf1
    printk 0,kf2
    kgoto nxtmsg
ex:
    endin
```

## Crédits

Auteur : John ffitich  
2005

# OSClisten

OSClisten — Ecoute les messages OSC sur un chemin particulier.

## Description

Cherche à chaque cycle-k si un message OSC a été envoyé à un certain chemin d'un certain type.

## Syntaxe

```
kans OSClisten ihandle, idest, itype [, xdata1, xdata2, ...]
```

## Initialisation

*ihandle* -- un identifiant retourné par un appel antérieur à *OSCinit*, pour associer *OSClisten* avec un numéro de port particulier.

*idest* -- une chaîne de caractères représentant l'adresse de destination. Elle est formatée comme un nom de fichier avec des répertoires. Csound utilise cette adresse pour décider si les messages sont destinés à Csound.

*itype* -- une chaîne de caractères indiquant le type des arguments optionnels à lire. La chaîne peut contenir les caractères "cdfhis" qui signifient caractère, double, flottant, entier sur 64 bit, entier sur 32 bit et chaîne de caractères. Tous les types sauf 's' nécessitent une variable de taux-k, tandis que 's' nécessite une variable chaîne de caractères.

Un identifiant est inséré dans le module d'écoute (voir *OSCinit*) pour intercepter les messages conformes à ce modèle.

## Exécution

*kans* -- fixé à 1 si un nouveau message a été reçu, ou 0 dans le cas contraire. Si plusieurs messages sont reçus dans une seule période de contrôle, les messages sont mis dans un tampon, et *OSClisten* peut être rappelé jusqu'à ce que 0 soit retourné.

S'il y avait un message les variables *xdata* reçoivent les valeurs en entrée, selon l'interprétation du paramètre *itype*. Noter que bien que les variables *xdata* soient situées à droite de l'opérateur, ce sont des sorties, et elles doivent donc être des variables k, gk, S ou gS, et peut-être nécessiter une déclaration avec *init* ou = dans le cas des variables chaîne de caractères, avant l'appel à *OSClisten*.

## Exemple

L'exemple montre une paire de nombres en virgule flottante reçus sur le port 7770.

```
sr = 44100
ksmps = 100
nchnls = 2

gihandle OSCinit 7770

instr 1
  kf1 init 0
  kf2 init 0
```

```
nxtmsg:
    kk  OSClisten  gihandle, "/foo/bar", "ff", kf1, kf2
if (kk == 0) goto ex
    printk 0,kf1
    printk 0,kf2
    kgoto nxtmsg
ex:
    endin
```

Ci-dessous deux fichiers .csd démontrent l'utilisation des opcodes OSC. Ils utilisent les fichiers *OSCmidisend.csd* [examples/OSCmidisend.csd] et *OSCmidircv.csd* [examples/OSCmidircv.csd].

### Exemple 338. Exemples des opcodes OSC.

Les deux fichiers .csd suivants démontrent l'utilisation des opcodes OSC dans Csound. Le premier fichier, *OSCmidisend.csd* [examples/OSCmidisend.csd], transforme des messages MIDI reçus en temps-réel en données OSC. Le second fichier, *OSCmidircv.csd* [examples/OSCmidircv.csd], peut prendre ces messages OSC et les interpréter pour générer du son à partir des messages de note, et stocker les valeurs de contrôleur. Il utilise le contrôleur 7 pour modifier le volume. Noter que ces fichiers sont conçus pour se trouver sur la même machine, mais si une adresse d'hôte différente (dans la macro IPADDRESS) est utilisée, ils peuvent se trouver sur différentes machines d'un réseau, ou connectées via l'internet.

Fichier CSD pour envoyer des messages OSC :

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

    sr      = 44100
    ksmpr   = 128
    nchnls  = 1

; Example by David Akbari 2007
; Modified by Jonathan Murphy
; Use this file to generate OSC events for OSCmidircv.csd

#define IPADDRESS # "localhost" #
#define PORT      # 47120 #

turnon 1000

    instr 1000

    kst, kch, kd1, kd2  midiin

    OSCsend  kst+kch+kd1+kd2, $IPADDRESS, $PORT, "/midi", "iiii", kst, kch, kd1, kd2

    endin

</CsInstruments>
<CsScore>
f 0 3600 ;Dummy f-table
e
</CsScore>
</CsoundSynthesizer>
```

Fichier CSD pour recevoir des messages OSC :

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

    sr      = 44100
    ksmps   = 128
    nchnls  = 1

; Example by Jonathan Murphy and Andres Cabrera 2007
; Use file OSCmidisend.csd to generate OSC events for this file

    odbfs    = 1

gilisten  OSCinit  47120

gisin      ftgen    1, 0, 16384, 10, 1
givel      ftgen    2, 0, 128, -2, 0
gicc       ftgen    3, 0, 128, -7, 100, 128, 100 ;Default all controllers to 100

;Define scale tuning
giji_12    ftgen    202, 0, 32, -2, 12, 2, 256, 60, 1, 16/15, 9/8, 6/5, 5/4, 4/3, 7/5, \
            3/2, 8/5, 5/3, 9/5, 15/8, 2

#define DEST #"/midi"#
; Use controller number 7 for volume
#define VOL #7#

turnon 1000

    instr 1000

    kst      init    0
    kch      init    0
    kdl      init    0
    kd2      init    0

next:

    kk      OSClisten gilisten, $DEST, "iiii", kst, kch, kdl, kd2

if (kk == 0) goto done

printks "kst = %i, kch = %i, kdl = %i, kd2 = %i\\n", \
        0, kst, kch, kdl, kd2

if (kst == 176) then
;Store controller information in a table
        tablew    kd2, kdl, gicc
endif

if (kst == 144) then
;Process noteon and noteoff messages.
    kkey      = kdl
    kvel      = kd2
    kcps      cpstun    kvel, kkey, giji_12
    kamp      = kvel/127

if (kvel == 0) then
        turnoff2 1001, 4, 1
elseif (kvel > 0) then
        event     "i", 1001, 0, -1, kcps, kamp
endif
endif

        kgoto next ;Process all events in queue

done:
    endin

    instr 1001 ;Simple instrument

    icps      init    p4
    kvol      table    $VOL, gicc ;Read MIDI volume from controller table
    kvol      = kvol/127

```

```
aenv    linsegr    0, .003, p5, 0.03, p5 * 0.5, 0.3, 0
aosc    oscil      aenv, icps, gisin

        out        aosc * kvol

    endin

</CsInstruments>
<CsScore>
f 0 3600 ;Dummy f-table
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch

2005

Exemples par David Akbari, Andrés Cabrera et Jonathan Murphy 2007



# oscbnk

oscbnk — Mélange la sortie de n'importe quel nombre d'oscillateurs.

## Description

Ce générateur unitaire mélange la sortie de n'importe quel nombre d'oscillateurs. La fréquence, la phase et l'amplitude de chaque oscillateur peuvent être modulées par deux LFO (tous les oscillateurs ont un jeu de LFO séparé, avec différentes phase et fréquence) ; de plus, la sortie de chaque oscillateur peut être filtrée au travers d'un égaliseur paramétrique (aussi contrôlé par les LFO). Cet opcode trouve sa plus grande utilité dans des instruments de rendu d'ensemble (cordes, cœur, etc.).

Bien que les LFO fonctionnent au taux-k, les modulations d'amplitude, de phase et de filtrage sont interpolées en interne, et il est ainsi possible (et recommandé dans la plupart des cas) d'utiliser cette unité avec de faibles taux de contrôle (~1000 Hz) sans dégradation audible de la qualité.

La phase et la fréquence initiale de tous les oscillateurs et LFO peuvent être fixées par un générateur intégré de nombres aléatoires sur 31 bit amorçable par une « graine », ou spécifiées manuellement dans une table de fonction (GEN2).

## Syntaxe

```
ares oscbnk kcps, kamd, kfmd, kpmd, iovrlap, iseed, kllminf, kllmaxf, \
    kl2minf, kl2maxf, ilfomode, kegminf, kegmaxf, kegminl, kegmaxl, \
    kegming, kegmaxq, iegmode, kfn [, il1fn] [, il2fn] [, iegffn] \
    [, ieglfm] [, iegqfn] [, itabl] [, ioutfn]
```

## Initialisation

*iovrlap* -- Nombre d'oscillateurs.

*iseed* -- Valeur de la graine du générateur de nombres aléatoires (entier positif dans l'intervalle 1 à 2147483646 ( $2^{31} - 2$ )). Si *iseed* <= 0 la graine est l'heure courante.

*iegmode* -- Mode de l'égaliseur paramétrique

- -1 : désactive l'EQ (plus rapide)
- 0 : crête
- 1 : à plateau low shelf
- 2 : à plateau high shelf
- 3 : crête (filtrage sans interpolation)
- 4 : à plateau low shelf (sans interpolation)
- 5 : à plateau high shelf (sans interpolation)

Les modes sans interpolation sont plus rapides, et dans certains cas (par exemple filtre à plateau high shelf aux fréquences de coupure basses) également plus stables ; cependant, l'interpolation est utile pour éviter le « bruit de fermeture éclair » aux faibles taux de contrôle.

*ilfomode* -- Type de la modulation par les LFO, somme de :

- 128 : LFO1 module la fréquence
- 64 : LFO1 module l'amplitude
- 32 : LFO1 module la phase
- 16 : LFO1 module l'EQ
- 8 : LFO2 module la fréquence
- 4 : LFO2 module l'amplitude
- 2 : LFO2 module la phase
- 1 : LFO2 module l'EQ

Si un LFO ne module rien, il n'est pas calculé, et le numéro de sa ftable (*il1fn* ou *il2fn*) peut être omis.

*il1fn* (facultatif : par défaut 0) -- Numéro de la table de fonction de LFO1. La forme d'onde dans cette table doit être normalisée (valeur absolue  $\leq 1$ ), et elle est lue avec une interpolation linéaire.

*il2fn* (facultatif : par défaut 0) -- Numéro de la table de fonction de LFO2. La forme d'onde dans cette table doit être normalisée (valeur absolue  $\leq 1$ ), et elle est lue avec une interpolation linéaire.

*ieqffn*, *ieqlfn*, *ieqqfn* (facultatif : par défaut 0) -- Tables de lecture pour la fréquence, le niveau et le Q de EQ (facultatif si EQ est désactivé). La position de lecture dans une table est 0 si le signal de modulation est inférieur ou égal à -1, (longueur de table / 2) si le signal de modulation vaut zero, et le point de garde si le signal de modulation est supérieur ou égal à 1. Ces tables doivent être normalisées dans l'intervalle 0 - 1, et ont un point de garde étendu (longueur de table = puissance de deux + 1). Toutes les tables sont lues avec une interpolation linéaire.

*itabl* (facultatif : par défaut 0) -- Table de fonction stockant les valeurs de phase et de fréquence pour tous les oscillateurs (facultatif). Les valeurs dans cette table sont dans l'ordre suivant (5 pour chaque oscillateur) :

phase de l'oscillateur, phase de lfo1, fréquence de lfo1, phase de lfo2, fréquence de lfo2, ...

Toutes les valeurs sont dans l'intervalle 0 à 1 ; si le nombre spécifié est supérieur à 1, il est ramené cycliquement (phase) ou limité (fréquence) à l'intérieur de l'intervalle permis. Une valeur négative (ou la fin de la table) utilisera la sortie du générateur de nombres aléatoires. La valeur aléatoire est toujours calculée (même si aucun nombre aléatoire n'est utilisé), si bien que le fait de basculer entre une valeur aléatoire et une valeur fixe n'altérera pas les autres valeurs.

*ioutfn* (facultatif : par défaut 0) -- Table de fonction pour écrire les valeurs de phase et de fréquence (facultatif). Le format est le même que celui de *itabl*. Cette table est utile lors de l'expérimentation avec des nombres aléatoires pour enregistrer les meilleures valeurs.

L'accès aux deux tables facultatives (*itabl* et *ioutfn*) n'a lieu que pendant l'initialisation. Il est utile de savoir cela, car les tables peuvent être réécrites en toute sécurité après l'initialisation de l'opcode, permettant le pré-calcul des paramètres pendant le temps-i et le stockage dans une table temporaire avant l'initialisation de *oscbnk*.

## Exécution

*ares* -- Signal de sortie.

*kcps* -- Fréquence de l'oscillateur en Hz.

*kamd* -- Profondeur de la modulation d'amplitude (0 - 1).

(sortie MA) = (entrée MA) \* ((1 - (prof MA)) + (prof MA) \* (modulateur))

Si *ilfomode* n'est pas réglé pour moduler l'amplitude, alors (sortie MA) = (entrée MA) quelque soit la valeur de *kamd*. Dans ce cas, *kamd* n'aura pas d'effet.

Note : La modulation d'amplitude est appliquée avant l'égaliseur paramétrique.

*kfmd* -- Profondeur de la MF (en Hz).

*kpm* -- Profondeur de la modulation de phase.

*kl1minf*, *kl1maxf* -- Fréquence minimale et maximale de LFO1 en Hz.

*kl2minf*, *kl2maxf* -- Fréquence minimale et maximale de LFO2 en Hz. (Note : il est permis d'avoir des fréquences nulles ou négatives pour l'oscillateur et les LFO.)

*keqminf*, *keqmaxf* -- Fréquence minimale et maximale de l'égaliseur paramétrique en Hz.

*keqminl*, *keqmaxl* -- Niveau minimum et maximum de l'égaliseur paramétrique.

*keqminq*, *keqmaxq* -- Q minimum et maximum de l'égaliseur paramétrique.

*kfn* -- Table de la forme d'onde de l'oscillateur. Le numéro de la table peut être changé au taux-k (c'est utile pour choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter les erreurs de repliement). La table est lue avec une interpolation linéaire.



## Note

*oscbnk* utilise le même générateur de nombres aléatoires que *rnd31*. C'est pourquoi il est également recommandé de lire *sa documentation*.

## Exemples

Voici un exemple de l'opcode *oscbnk*. Il utilise le fichier *oscbnk.csd* [examples/oscbnk.csd].

### Exemple 339. Exemple de l'opcode *oscbnk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscbnk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr = 48000
```

```

kr = 750
ksmps = 64
nchnls = 2

ga01 init 0
ga02 init 0

/* sawtooth wave */
i_ ftgen 1, 0, 16384, 7, 1, 16384, -1
/* FM waveform */
i_ ftgen 3, 0, 4096, 7, 0, 512, 0.25, 512, 1, 512, 0.25, 512, \
    0, 512, -0.25, 512, -1, 512, -0.25, 512, 0
/* AM waveform */
i_ ftgen 4, 0, 4096, 5, 1, 4096, 0.01
/* FM to EQ */
i_ ftgen 5, 0, 1024, 5, 1, 512, 32, 512, 1
/* sine wave */
i_ ftgen 6, 0, 1024, 10, 1
/* room parameters */
i_ ftgen 7, 0, 64, -2, 4, 50, -1, -1, -1, 11, \
    1, 26.833, 0.05, 0.85, 10000, 0.8, 0.5, 2, \
    1, 1.753, 0.05, 0.85, 5000, 0.8, 0.5, 2, \
    1, 39.451, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 33.503, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 36.151, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 29.633, 0.05, 0.85, 7000, 0.8, 0.5, 2

/* generate bandlimited sawtooth waves */

i0 = 0
loop1:
imaxh = sr / (2 * 440.0 * exp (log(2.0) * (i0 - 69) / 12))
i_ ftgen i0 + 256, 0, 4096, -30, 1, 1, imaxh
i0 = i0 + 1
    if (i0 < 127.5) igoto loop1

instr 1

p3 = p3 + 0.4

; note frequency
kcps = 440.0 * exp (log(2.0) * (p4 - 69) / 12)
; lowpass max. frequency
klpmaxf limit 64 * kcps, 1000.0, 12000.0
; FM depth in Hz
kfmd1 = 0.02 * kcps
; AM frequency
kamfr = kcps * 0.02
kamfr2 = kcps * 0.1
; table number
kfnum = (256 + 69 + 0.5 + 12 * log(kcps / 440.0) / log(2.0))
; amp. envelope
aenv linseg 0, 0.1, 1.0, p3 - 0.5, 1.0, 0.1, 0.5, 0.2, 0, 1.0, 0

/* oscillator / left */

a1 oscbnk kcps, 0.0, kfmd1, 0.0, 40, 200, 0.1, 0.2, 0, 0, 144, \
    0.0, klpmaxf, 0.0, 0.0, 1.5, 1.5, 2, \
    kfnum, 3, 0, 5, 5, 5
a2 oscbnk kcps, 1.0, kfmd1, 0.0, 40, 201, 0.1, 0.2, kamfr, kamfr2, 148, \
    0, 0, 0, 0, 0, 0, -1, \
    kfnum, 3, 4
a2 pareq a2, kcps * 8, 0.0, 0.7071, 2
a0 = a1 + a2 * 0.12
/* delay */
adel = 0.001
a01 vdelayx a0, adel, 0.01, 16
a_ oscili 1.0, 0.25, 6, 0.0
adel = adel + 1.0 / (exp(log(2.0) * a_) * 8000)
a02 vdelayx a0, adel, 0.01, 16
a0 = a01 + a02

ga01 = ga01 + a0 * aenv * 2500

/* oscillator / right */

; lowpass max. frequency

a1 oscbnk kcps, 0.0, kfmd1, 0.0, 40, 202, 0.1, 0.2, 0, 0, 144, \
    0.0, klpmaxf, 0.0, 0.0, 1.0, 1.0, 2, \
    kfnum, 3, 0, 5, 5, 5

```

```

a2 oscbnk kcps, 1.0, kfmd1, 0.0, 40, 203, 0.1, 0.2, kamfr, kamfr2, 148, \
      0, 0, 0, 0, 0, 0, -1, \
      kfnum, 3, 4
a2 pareq a2, kcps * 8, 0.0, 0.7071, 2
a0 = a1 + a2 * 0.12
/* delay */
adel = 0.001
a01 vdelayx a0, adel, 0.01, 16
a_ oscili 1.0, 0.25, 6, 0.25
adel = adel + 1.0 / (exp(log(2.0) * a_) * 8000)
a02 vdelayx a0, adel, 0.01, 16
a0 = a01 + a02

ga02 = ga02 + a0 * aenv * 2500

      endin

/* output / left */

      instr 81

i1 = 0.000001
aLl, aLh, aRl, aRh spat3di ga01 + i1*i1*i1*i1, -8.0, 4.0, 0.0, 0.3, 7, 4
ga01 = 0
aLl butterlp aLl, 800.0
aRl butterlp aRl, 800.0

      outs aLl + aLh, aRl + aRh

      endin

/* output / right */

      instr 82

i1 = 0.000001
aLl, aLh, aRl, aRh spat3di ga02 + i1*i1*i1*i1, 8.0, 4.0, 0.0, 0.3, 7, 4
ga02 = 0
aLl butterlp aLl, 800.0
aRl butterlp aRl, 800.0

      outs aLl + aLh, aRl + aRh

      endin

</CsInstruments>
<CsScore>

/* Written by Istvan Varga */
t 0 60

i 1 0 4 41
i 1 0 4 60
i 1 0 4 65
i 1 0 4 69

i 81 0 5.5
i 82 0 5.5
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga  
2001

Nouveau dans la version 4.15

Mis à jour en avril 2002 par Istvan Varga

# oscil

oscil — Un oscillateur simple.

## Description

*oscil* lit la table *ifn* séquentiellement et de manière répétitive à la fréquence *xcps*. L'amplitude est pondérée par *xamp*.

## Syntaxe

```
ares oscil xamp, xcps, ifn [ , iphs]
```

```
kres oscil kamp, kcps, ifn [ , iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique.

*iphs* (facultatif, par défaut 0) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*kamp*, *xamp* -- amplitude

*kcps*, *xcps* -- fréquence en cycles par seconde.

L'opcode *oscil* génère des signaux de contrôle (ou audio) constitués de la valeur de *kamp* (*xamp*) fois la valeur de la lecture au taux de contrôle (ou au taux audio) d'une table de fonction stockée. La phase interne est simultanément incrémentée selon la valeur en entrée de *kcps* ou de *xcps*.

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Si vous désirez changer la table de l'oscillateur avec un signal de taux-k, vous pouvez utiliser *oscilikt*.

## Exemples

Voici un exemple de l'opcode *oscil*. Il utilise le fichier *oscil.csd* [examples/oscil.csd].

### Exemple 340. Exemple de l'opcode *oscil*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 128
nchnls = 2

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  a1 oscil kamp, kcps, ifn
  outs a1,a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*oscili, oscilikt, oscil3, poscil, poscil3*

## Crédits

Exemple écrit par Kevin Conder.

# oscil1

oscil1 — Accède aux valeurs d'une table par échantillonnage incrémentiel.

## Description

Accède aux valeurs d'une table par échantillonnage incrémentiel.

## Syntaxe

```
kres oscil1 idel, kamp, idur, ifn
```

## Initialisation

*idel* -- délai en secondes avant que l'échantillonnage incrémentiel d'*oscil1* ne commence.

*idur* -- durée en secondes de l'unique passe d'échantillonnage dans la table d'*oscil1*. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

*ifn* -- numéro de la table de fonction. *tablei*, *oscilli* nécessitent un point de garde.

## Exécution

*kamp* -- facteur d'amplitude.

*oscil1* accède aux valeurs en échantillonnant une fois la table de fonction à un taux déterminé par *idur*. Pendant les premières *idel* secondes, le point de lecture reste sur la première position de la table ; ensuite il traverse la table à vitesse constante, atteignant la fin au bout de *idur* secondes ; à partir de ce moment (c-à-d après *idel* + *idur* secondes) il reste sur la dernière position. Chaque valeur lue par échantillonnage est multipliée par le facteur d'amplitude *kamp* avant d'être écrite dans le résultat.

## Voir Aussi

*table*, *tablei*, *table3*, *oscilli*, *osciln*



# oscil1i

oscil1i — Accède aux valeurs d'une table par échantillonnage incrémentiel avec interpolation linéaire.

## Description

Accède aux valeurs d'une table par échantillonnage incrémentiel avec interpolation linéaire.

## Syntaxe

```
kres oscil1i idel, kamp, idur, ifn
```

## Initialisation

*idel* -- délai en secondes avant que l'échantillonnage incrémentiel d'*oscil1i* ne commence.

*idur* -- durée en secondes de l'unique passe d'échantillonnage dans la table d'*oscil1i*. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

*ifn* -- numéro de la table de fonction. *oscil1i* nécessitent un point de garde.

## Exécution

*kamp* -- facteur d'amplitude

*oscil1i* est une unité avec interpolation dans laquelle la partie fractionnaire de l'index est utilisée pour interpoler entre les entrées adjacentes de la table. La régularité apportée par l'interpolation se paie par une légère augmentation du temps d'exécution (voir aussi *oscili*, etc.), mais sinon les unités avec ou sans interpolation sont interchangeables.

## Voir Aussi

*table*, *tablei*, *table3*, *oscil1*, *osciln*

# oscil3

oscil3 — Un oscillateur simple avec interpolation cubique.

## Description

*oscil3* lit la table *ifn* séquentiellement et de manière répétitive à la fréquence *xcps*. L'amplitude est pondérée par *xamp*. La lecture des valeurs de phase internes de la table se fait avec interpolation cubique.

## Syntaxe

```
ares oscil3 xamp, xcps, ifn [, iphs]
```

```
kres oscil3 kamp, kcps, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique.

*iphs* (facultatif) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*kamp*, *xamp* -- amplitude

*kcps*, *xcps* -- fréquence en cycles par seconde.

*oscil3* est identique à *oscili*, sauf qu'il utilise l'interpolation cubique.

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Si vous désirez changer la table de l'oscillateur avec un signal de taux-k, vous pouvez utiliser *oscilikt*.

## Exemples

Voici un exemple de l'opcode *oscil3*. Il utilise le fichier *oscil3.csd* [examples/oscil3.csd].

### Exemple 341. Exemple de l'opcode *oscil3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac       -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o oscil3.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
    kamp = 10000
    kcps = 220
    ifn = 1

    a1 oscil kamp, kcps, ifn
    out a1
endin

; Instrument #2 - the basic oscillator with cubic interpolation.
instr 2
    kamp = 10000
    kcps = 220
    ifn = 1

    a1 oscil3 kamp, kcps, ifn
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave table with a small amount of data.
f 1 0 32 10 0 1

; Play Instrument #1, the basic oscillator, for
; two seconds. This should sound relatively rough.
i 1 0 2

; Play Instrument #2, the cubic interpolated oscillator, for
; two seconds. This should sound relatively smooth.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*oscil, oscili, oscilikt.*

## Crédits

Auteur : John ffitich

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.50 de Csound

# oscili

oscili — Un oscillateur simple avec interpolation linéaire.

## Description

*oscili* lit la table *ifn* séquentiellement et de manière répétitive à la fréquence *xcps*. L'amplitude est pondérée par *xamp*. La lecture des valeurs de phase internes de la table se fait avec interpolation linéaire.

## Syntaxe

```
ares oscili xamp, xcps, ifn [, iphs]
```

```
kres oscili kamp, kcps, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique.

*iphs* (facultatif) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*kamp*, *xamp* -- amplitude

*kcps*, *xcps* -- fréquence en cycles par seconde.

*oscili* diffère de *oscil* en ce que la procédure standard d'utilisation d'une phase tronquée comme index de lecture est remplacée ici par une interpolation entre deux lectures successives. Les générateurs avec interpolation produiront un signal de sortie nettement plus propre, mais ils peuvent prendre jusqu'à deux fois plus de temps de calcul. On peut obtenir également ce type de précision sans le surcoût du calcul de l'interpolation en utilisant de grandes tables de fonction stockées de 2K, 4K ou 8K points, si l'on dispose de cet espace mémoire.

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Si vous désirez changer la table de l'oscillateur avec un signal de taux-k, vous pouvez utiliser *oscilikt*.

## Exemples

Voici un exemple de l'opcode *oscili*. Il utilise le fichier *oscili.csd* [examples/oscili.csd].

### Exemple 342. Exemple de l'opcode *oscili*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscili.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 220
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin

; Instrument #2 - the basic oscillator with extra interpolation.
instr 2
  kamp = 10000
  kcps = 220
  ifn = 1

  a1 oscili kamp, kcps, ifn
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave table with a small amount of data.
f 1 0 32 10 0 1

; Play Instrument #1, the basic oscillator, for
; two seconds. This should sound relatively rough.
i 1 0 2

; Play Instrument #2, the interpolated oscillator, for
; two seconds. This should sound relatively smooth.
i 2 2 2
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir Aussi

*oscil, oscil3*

## Crédits

Exemple écrit par Kevin Conder.

# oscilikt

oscilikt — Un oscillateur avec interpolation linéaire qui permet de changer le numéro de table au taux-k.

## Description

*oscilikt* ressemble beaucoup à *oscili*, mais il permet de changer le numéro de table au taux-k. Il est légèrement plus lent que *oscili* (spécialement avec des taux de contrôle élevés), mais en contrepartie il est plus précis car il utilise un accumulateur de phase sur 31 bit au lieu de celui sur 24 bit utilisé par *oscili*.

## Syntaxe

```
ares oscilikt xamp, xcps, kfn [, iphs] [, istor]
```

```
kres oscilikt kamp, kcps, kfn [, iphs] [, istor]
```

## Initialisation

*iphs* (facultatif, par défaut 0) -- phase initiale dans l'intervalle 0 à 1. Les autres valeurs sont ramenées cycliquement dans l'intervalle autorisé.

*istor* (facultatif, par défaut 0) -- ignorer l'initialisation.

## Exécution

*kamp*, *xamp* -- amplitude.

*kcps*, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à *sr/2*).

*kfn* -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par *GEN30*).

## Exemples

Voici un exemple de l'opcode *oscilikt*. Il utilise le fichier *oscilikt.csd* [examples/oscilikt.csd].

### Exemple 343. Exemple de l'opcode *oscilikt*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscilikt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a uni-polar (0-1) square wave.
kamp1 init 1
kcps1 init 2
itype = 3
ksquare lfo kamp1, kcps1, itype

; Use the square wave to switch between Tables #1 and #2.
kamp2 init 20000
kcps2 init 220
kfn = ksquare + 1

a1 oscilikt kamp2, kcps2, kfn
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine waveform.
f 1 0 4096 10 0 1
; Table #2: a sawtooth wave
f 2 0 3 -2 1 0 -1

; Play Instrument #1 for two seconds.
i 1 0 2

</CsScore>
</CsSoundSynthesizer>

```

## Voir Aussi

*osciliktp* et *oscilikts*.

## Crédits

Auteur : Istvan Varga

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.22

# osciliktp

osciliktp — Un oscillateur avec interpolation linéaire qui permet la modulation de phase.

## Description

*osciliktp* permet la modulation de phase (qui est implémentée comme une modulation de fréquence au taux-*k*, en différenciant la phase en entrée). Le désavantage est qu'il n'y a pas de contrôle d'amplitude, et que la fréquence ne peut varier qu'au taux de contrôle. Cet opcode peut être plus rapide ou plus lent que *oscilikt*, en fonction du taux de contrôle.

## Syntaxe

```
ares osciliktp kcps, kfn, kphs [, istor]
```

## Initialisation

*istor* (facultatif, par défaut 0) -- ignorer l'initialisation.

## Exécution

*ares* -- signal de sortie au taux audio.

*kcps*, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à *sr/2*).

*kfn* -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par *GEN30*).

*kphs* -- phase (taux-*k*), l'intervalle attendu est 0 à 1. La valeur absolue de la différence entre les valeurs courante et précédente de *kphs* doit être inférieure à *ksmps*.

## Exemples

Voici un exemple de l'opcode *osciliktp*. Il utilise le fichier *osciliktp.csd* [examples/osciliktp.csd].

### Exemple 344. Exemple de l'opcode *osciliktp*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o osciliktp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```



```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1: oscilikt example
instr 1
  kphs line 0, p3, 4

  alx oscilikt 220.5, 1, 0
  aly oscilikt 220.5, 1, -kphs
  al = alx - aly

  out al * 14000
endin

</CsInstruments>
<CsScore>

; Table #1: Sawtooth wave
f 1 0 3 -2 1 0 -1

; Play Instrument #1 for four seconds.
i 1 0 4
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*oscilikt* et *oscilikts*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# oscilikts

oscilikts — Un oscillateur avec interpolation linéaire et statut de synchronisation qui permet de changer le numéro de table au taux-k.

## Description

*oscilikts* est pareil à *oscilikt*. Sauf qu'il a une entrée de synchronisation que l'on peut utiliser pour réinitialiser l'oscillateur à une valeur de phase de taux-k. Il est plus lent que *oscilikt* et que *osciliktp*.

## Syntaxe

```
ares oscilikts xamp, xcps, kfn, async, kphs [, istor]
```

## Initialisation

*istor* (facultatif, par défaut 0) -- ignorer l'initialisation.

## Exécution

*xamp* -- amplitude.

*kcps*, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à  $sr/2$ ).

*kfn* -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par GEN30).

*async* -- n'importe quelle valeur positive réinitialise la valeur de la phase de *oscilikts* à *kphs*. Zero ou des valeurs négatives n'ont aucun effet.

*kphs* -- fixe la phase, initialement et lorsqu'elle est réinitialisée avec *async*.

## Exemples

Voici un exemple de l'opcode *oscilikts*. Il utilise le fichier *oscilikts.csd* [examples/oscilikts.csd].

### Exemple 345. Exemple de l'opcode *oscilikts*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac       -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o oscilikts.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1: oscilikt example.
instr 1
; Frequency envelope.
kfrq expon 400, p3, 1200
; Phase.
kphs line 0.1, p3, 0.9

; Sync 1
atmp1 phasor 100
; Sync 2
atmp2 phasor 150
async diff 1 - (atmp1 + atmp2)

a1 oscilikt 14000, kfrq, 1, async, 0
a2 oscilikt 14000, kfrq, 1, async, -kphs

out a1 - a2
endin

</CsInstruments>
<CsScore>

; Table #1: Sawtooth wave
f 1 0 3 -2 1 0 -1

; Play Instrument #1 for four seconds.
i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*oscilikt* et *osciliktp*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# osciln

osciln — Lit des valeurs dans une table à une fréquence définie par l'utilisateur.

## Description

Lit des valeurs dans une table à une fréquence définie par l'utilisateur. On peut également écrire cet opcode comme *oscilx*.

## Syntaxe

```
ares osciln kamp, ifrq, ifn, itimes
```

## Initialisation

*ifrq, itimes* -- taux de lecture et nombre de passages à travers la table.

*ifn* -- numéro de la table de fonction.

## Exécution

*kamp* -- facteur d'amplitude

*osciln* parcourera plusieurs fois la table stockée en prélevant un échantillon *ifrq* fois par seconde, après quoi il retournera des zéros. Il génère seulement des signaux audio, avec les valeurs de sortie pondérées par *kamp*.

## Voir aussi

*table, tablei, table3, oscill, oscilli*

# oscils

oscils — Un oscillateur sinus simple et rapide.

## Description

Oscillateur sinus simple et rapide, qui utilise seulement une multiplication et deux additions pour générer un échantillon en sortie, et qui ne nécessite pas de table de fonction.

## Syntaxe

```
ares oscils iamp, icps, iphs [, iflg]
```

## Initialisation

*iamp* -- amplitude en sortie.

*icps* -- fréquence en Hz (peut être nulle ou négative, cependant la valeur absolue doit être inférieure à  $sr/2$ ).

*iphs* -- phase initiale entre 0 et 1.

*iflg* -- somme des valeurs suivantes :

- 2 : utiliser la double précision même si Csound a été compilé pour utiliser des floats. Ceci améliore la qualité (spécialement dans le cas d'une longue exécution), mais le temps de calcul peut varier du simple au double.
- 1 : ignorer l'initialisation.

## Exécution

*ares* -- sortie audio

## Exemples

Voici un exemple de l'opcode oscils. Il utilise le fichier *oscils.csd* [examples/oscils.csd].

### Exemple 346. Exemple de l'opcode oscils.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscils.wav -W ;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a fast sine oscillator.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  a1 oscils iamp, icps, iphs
  out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Istvan Varga  
Janvier 2002

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.18

## oscilx

oscilx — Identique à l'opcode osciln.

## Description

Voir l'opcode *osciln*.

# out

out — Ecrit des données audio mono vers un périphérique externe ou un flot.

## Description

Ecrit des données audio mono vers un périphérique externe ou un flot.

## Syntaxe

`out asig`

## Exécution

Envoie des échantillons audio mono dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*.

## Voir Aussi

*outh, outho, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

Original dans Csound v1



# out32

out32 — Ecrit des données audio sur 32 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 32 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
out32 asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, asig10, \  
      asig11, asig12, asig13, asig14, asig15, asig16, asig17, asig18, \  
      asig19, asig20, asig21, asig22, asig23, asig24, asig25, asig26, \  
      asig27, asig28, asig29, asig30, asig31, asig32
```

## Exécution

*out32* sort 32 canaux d'audio.

## Voir Aussi

*outc, outch, outx, outz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# outc

*outc* — Ecrit des données audio sur un nombre arbitraire de canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur un nombre arbitraire de canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outc asig1 [, asig2] [...]
```

## Exécution

*outc* écrit autant de canaux que de variables fournies. Tous les canaux dépassant *nchnls* sont ignorés. Des zéros sont ajoutés si nécessaire.

## Voir Aussi

*out32*, *outch*, *outx*, *outz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# outch

**outch** — Ecrit des données audio multi-canaux sous contrôle de l'utilisateur, vers un périphérique externe ou un flot.

## Description

Ecrit des données audio multi-canaux sous contrôle de l'utilisateur, vers un périphérique externe ou un flot.

## Syntaxe

```
outch kchan1, asig1 [, kchan2] [, asig2] [...]
```

## Exécution

*outch* envoie *asig1* sur le canal déterminé par *kchan1*, *asig2* sur le canal déterminé par *kchan2*, etc.



### Note

Le plus grand numéro de paramètre *kchanX* pour *outch* dépend de *nchnls*. Si *kchanX* est supérieur à *nchnls*, *asigX* sera silencieux. Noter que *outch* donnera dans ce cas un avertissement mais pas d'erreur.

## Voir Aussi

*out32*, *outc*, *outx*, *outz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# outh

outh — Ecrit des données audio sur 6 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 6 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outh asig1, asig2, asig3, asig4, asig5, asig6
```

## Exécution

Envoie des échantillons sur 6 canaux dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*.

## Voir Aussi

*out, outh, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteur : John fitch

Introduit avant la version 3

# outiat

outiat — Sends MIDI aftertouch messages at i-rate.

## Description

Sends MIDI aftertouch messages at i-rate.

## Syntax

```
outiat ichn, ivalue, imin, imax
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*ivalue* -- floating point value

*imin* -- minimum floating point value (converted in MIDI integer value 0)

*imax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

## Performance

*outiat* (i-rate aftertouch output) sends aftertouch messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

## See Also

*outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outic

outic — Sends MIDI controller output at i-rate.

## Description

Sends MIDI controller output at i-rate.

## Syntax

```
outic ichn, inum, ivalue, imin, imax
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*inum* -- controller number (0-127 for example 1 = ModWheel; 2 = BreathControl etc.)

*ivalue* -- floating point value

*imin* -- minimum floating point value (converted in MIDI integer value 0)

*imax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

## Performance

*outic* (i-rate MIDI controller output) sends controller messages to the MIDI OUT device. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

## See Also

*outiat*, *outic14*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outic14

outic14 — Sends 14-bit MIDI controller output at i-rate.

## Description

Sends 14-bit MIDI controller output at i-rate.

## Syntax

```
outic14 ichn, imsb, ilsb, ivalue, imin, imax
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*imsb* -- most significant byte controller number when using 14-bit parameters (0-127)

*ilsb* -- least significant byte controller number when using 14-bit parameters (0-127)

*ivalue* -- floating point value

*imin* -- minimum floating point value (converted in MIDI integer value 0)

*imax* -- maximum floating point value (converted in MIDI integer value 16383 (14-bit))

## Performance

*outic14* (i-rate MIDI 14-bit controller output) sends a pair of controller messages. This opcode can drive 14-bit parameters on MIDI instruments that recognize them. The first control message contains the most significant byte of *ivalue* argument while the second message contains the less significant byte. *imsb* and *ilsb* are the number of the most and less significant controller.

This opcode can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

## See Also

*outiat*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.



# outipat

outipat — Sends polyphonic MIDI aftertouch messages at i-rate.

## Description

Sends polyphonic MIDI aftertouch messages at i-rate.

## Syntax

```
outipat ichn, inotenum, ivalue, imin, imax
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*inotenum* -- MIDI note number (used in polyphonic aftertouch messages)

*ivalue* -- floating point value

*imin* -- minimum floating point value (converted in MIDI integer value 0)

*imax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

## Performance

*outipat* (i-rate polyphonic aftertouch output) sends polyphonic aftertouch messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

## See Also

*outiat*, *outic14*, *outic*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outipb

outipb — Sends MIDI pitch-bend messages at i-rate.

## Description

Sends MIDI pitch-bend messages at i-rate.

## Syntax

```
outipb ichn, ivalue, imin, imax
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*ivalue* -- floating point value

*imin* -- minimum floating point value (converted in MIDI integer value 0)

*imax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

## Performance

*outipb* (i-rate pitch bend output) sends pitch bend messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

## See Also

*outiat*, *outic14*, *outic*, *outipat*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outipc

outipc — Sends MIDI program change messages at i-rate

## Description

Sends MIDI program change messages at i-rate

## Syntax

```
outipc ichn, iprog, imin, imax
```

## Initialization

*ichn* -- MIDI channel number (1-16)

*iprog* -- program change number in floating point

*imin* -- minimum floating point value (converted in MIDI integer value 0)

*imax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

## Performance

*outipc* (i-rate program change output) sends program change messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale an i-value floating-point argument according to the *imin* and *imax* values. For example, set *imin* = 1.0 and *imax* = 2.0. When the *ivalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *ivalue* argument receives a 1.0 value, it will send a 0 value. i-rate opcodes send their message once during instrument initialization.

## See Also

*outiat*, *outic14*, *outic*, *outipat*, *outipb*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outkat

outkat — Sends MIDI aftertouch messages at k-rate.

## Description

Sends MIDI aftertouch messages at k-rate.

## Syntax

**outkat** *kchn*, *kvalue*, *kmin*, *kmax*

## Performance

*kchn* -- MIDI channel number (1-16)

*kvalue* -- floating point value

*kmin* -- minimum floating point value (converted in MIDI integer value 0)

*kmax* -- maximum floating point value (converted in MIDI integer value 127)

*outkat* (k-rate aftertouch output) sends aftertouch messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

## See Also

*outiat*, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outkc

outkc — Sends MIDI controller messages at k-rate.

## Description

Sends MIDI controller messages at k-rate.

## Syntax

**outkc** *kchn*, *knum*, *kvalue*, *kmin*, *kmax*

## Performance

*kchn* -- MIDI channel number (1-16)

*knum* -- controller number (0-127 for example 1 = ModWheel; 2 = BreathControl etc.)

*kvalue* -- floating point value

*kmin* -- minimum floating point value (converted in MIDI integer value 0)

*kmax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

*outkc* (k-rate MIDI controller output) sends controller messages to MIDI OUT device. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

## See Also

*outiat*, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outkc14

outkc14 — Sends 14-bit MIDI controller output at k-rate.

## Description

Sends 14-bit MIDI controller output at k-rate.

## Syntax

```
outkc14 kchn, kmsb, klsb, kvalue, kmin, kmax
```

## Performance

*kchn* -- MIDI channel number (1-16)

*kmsb* -- most significant byte controller number when using 14-bit parameters (0-127)

*klsb* -- least significant byte controller number when using 14-bit parameters (0-127)

*kvalue* -- floating point value

*kmin* -- minimum floating point value (converted in MIDI integer value 0)

*kmax* -- maximum floating point value (converted in MIDI integer value 16383 (14-bit))

*outkc14* (k-rate MIDI 14-bit controller output) sends a pair of controller messages. It works only with MIDI instruments which recognize them. These opcodes can drive 14-bit parameters on MIDI instruments that recognize them. The first control message contains the most significant byte of *kvalue* argument while the second message contains the less significant byte. *kmsb* and *klsb* are the number of the most and less significant controller.

It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

## See Also

*outiat*, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outkpat

outkpat — Sends polyphonic MIDI aftertouch messages at k-rate.

## Description

Sends polyphonic MIDI aftertouch messages at k-rate.

## Syntax

**outkpat** *kchn*, *knotenum*, *kvalue*, *kmin*, *kmax*

## Performance

*kchn* -- MIDI channel number (1-16)

*knotenum* -- MIDI note number (used in polyphonic aftertouch messages)

*kvalue* -- floating point value

*kmin* -- minimum floating point value (converted in MIDI integer value 0)

*kmax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

*outkpat* (k-rate polyphonic aftertouch output) sends polyphonic aftertouch messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

## See Also

*outiat*, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpb*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outkpb

outkpb — Sends MIDI pitch-bend messages at k-rate.

## Description

Sends MIDI pitch-bend messages at k-rate.

## Syntax

**outkpb** *kchn*, *kvalue*, *kmin*, *kmax*

## Performance

*kchn* -- MIDI channel number (1-16)

*kvalue* -- floating point value

*kmin* -- minimum floating point value (converted in MIDI integer value 0)

*kmax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

*outkpb* (k-rate pitch-bend output) sends pitch-bend messages. It works only with MIDI instruments which recognize them. It can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

## See Also

*outiat*, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpc*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.



# outkpc

outkpc — Sends MIDI program change messages at k-rate.

## Description

Sends MIDI program change messages at k-rate.

## Syntax

**outkpc** *kchn*, *kprog*, *kmin*, *kmax*

## Performance

*kchn* -- MIDI channel number (1-16)

*kprog* -- program change number in floating point

*kmin* -- minimum floating point value (converted in MIDI integer value 0)

*kmax* -- maximum floating point value (converted in MIDI integer value 127 (7 bit))

*outkpc* (k-rate program change output) sends program change messages. It works only with MIDI instruments which recognize them. These opcodes can drive a different value of a parameter for each note currently active.

It can scale the k-value floating-point argument according to the *kmin* and *kmax* values. For example: set *kmin* = 1.0 and *kmax* = 2.0. When the *kvalue* argument receives a 2.0 value, the opcode will send a 127 value to the MIDI OUT device. When the *kvalue* argument receives a 1.0 value, it will send a 0 value. k-rate opcodes send a message each time the MIDI converted value of argument *kvalue* changes.

## Examples

Here is an example of the outkpc opcode. It uses the file *outkpc.csd* [examples/outkpc.csd].

### Exemple 347. Example of the outkpc opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example generates a program change and a note on Csound's MIDI output port whenever a note is received on channel 1. Be sure to have something connected to Csound's MIDI out port to hear the result.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadac    -d          -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
```

```

kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

kprogram init 0

instr 1 ;Triggered by MIDI notes on channel 1

    ifund    notnum
    ivel     veloc
    idur = 1

; Sends a MIDI program change message according to
; the triggering note's velocity
outkpc      1 ,ivel ,0 ,127

noteondur   1 ,ifund ,ivel ,idur

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>

```

Here is another example of the outkpc opcode. It uses the file *outkpc\_fltk.csd* [examples/outkpc\_fltk.csd].

### Exemple 348. Example of the outkpc opcode using FLTK.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d          -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

FLpanel "outkpc",200,100,90,90;start of container
gkpg, gihandle FLcount "Midi-Program change",0,127,1,5,1,152,40,16,23,-1
FLpanelEnd

FLrun

instr 1

ktrig changed gkpg
outkpc      ktrig,gkpg,0,127

endin

</CsInstruments>
<CsScore>
; Run instrument 1 for 60 seconds
i 1 0 60
</CsScore>
</CsoundSynthesizer>

```

## See Also

*outiat, outic14, outic, outipat, outipb, outipc, outkat, outkc14, outkc, outkpat, outkpb*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# outleta

outleta — Sends an arate signal out from an instrument to a named port.

## Description

Sends an arate signal out from an instrument to a named port.

## Syntax

`outleta Sname, asignal`

## Initialization

*Sname* -- String name of the outlet port. The name of the outlet is implicitly qualified by the instrument name or number, so it is valid to use the same outlet name in more than one instrument (but not to use the same outlet name twice in one instrument).

## Performance

*asignal* -- audio output signal

During performance, the audio output signal is sent to each instance of an instrument containing an inlet port to which this outlet has been connected using the connect opcode. The signals of all the outlets connected to an inlet are summed in the inlet.

## See Also

*outletk outletf inleta inletk inletf connect alwayson ftgenonce*

## Credits

By: Michael Gogins 2009

# outletk

outletk — Sends a krate signal out from an instrument to a named port.

## Description

Sends a krate signal out from an instrument to a named port.

## Syntax

`outletk Sname, ksignal`

## Initialization

*Sname* -- String name of the outlet port. The name of the outlet is implicitly qualified by the instrument name or number, so it is valid to use the same outlet name in more than one instrument (but not to use the same outlet name twice in one instrument).

## Performance

*ksignal* -- audio output signal

During performance, the output signal is sent to each instance of an instrument containing an inlet port to which this outlet has been connected using the connect opcode. The signals of all the outlets connected to an inlet are summed in the inlet.

## See Also

*outleta outletf inleta inletk inletf connect alwayson ftgenonce*

## Credits

By: Michael Gogins 2009

# outletf

outletf — Sends a frate signal (fsig) out from an instrument to a named port.

## Description

Sends a frate signal (fsig) out from an instrument to a named port.

## Syntax

```
outletf Sname, fsignal
```

## Initialization

*Sname* -- String name of the outlet port. The name of the outlet is implicitly qualified by the instrument name or number, so it is valid to use the same outlet name in more than one instrument (but not to use the same outlet name twice in one instrument).

## Performance

*fsignal* -- frate output signal (fsig)

During performance, the output signal is sent to each instance of an instrument containing an inlet port to which this outlet has been connected using the connect opcode. The signals of all the outlets connected to an inlet are combined in the inlet.

## See Also

*outleta outletk inleta inletk inletf connect alwayson ftgenonce*

## Credits

By: Michael Gogins 2009

# outo

outo — Ecrit des données audio sur 8 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 8 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outo asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8
```

## Exécution

Envoie des échantillons sur 8 canaux dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*.

## Voir Aussi

*out, outh, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteur : John ffitich

Nouveau après la 3.30

# outq

outq — Ecrit des données audio sur 4 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 4 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outq asig1, asig2, asig3, asig4
```

## Exécution

Envoie des échantillons sur 4 canaux dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadra n°3, etc.

## Voir Aussi

*out, outh, outo, outq1, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997



# outq1

outq1 — Ecrit des échantillons sur le canal quadro n°1 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons sur le canal quadro n°1 d'un périphérique externe ou d'un flot.

## Syntaxe

`outq1 asig`

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Voir Aussi

*out, outh, outh, outq, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

## outq2

outq2 — Ecrit des échantillons sur le canal quadro n°2 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons sur le canal quadro n°2 d'un périphérique externe ou d'un flot.

## Syntaxe

`outq2 asig`

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Voir Aussi

*out*, *outh*, *outo*, *outq*, *outq1*, *outq3*, *outq4*, *outs*, *outs1*, *outs2*, *soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

## outq3

outq3 — Ecrit des échantillons sur le canal quadro n°3 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons sur le canal quadro n°3 d'un périphérique externe ou d'un flot.

## Syntaxe

`outq3 asig`

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Voir Aussi

*out, outh, outho, outq, outq1, outq2, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

## outq4

outq4 — Ecrit des échantillons sur le canal quadro n°4 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons sur le canal quadro n°4 d'un périphérique externe ou d'un flot.

## Syntaxe

`outq4 asig`

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Voir Aussi

*out, outh, outh, outq, outq1, outq2, outq3, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# outrg

outrg — Permet la sortie dans un ensemble de canaux contigus sur le périphérique de sortie audio.

## Description

*outrg* sort les données audio dans un ensemble de canaux contigus sur le périphérique de sortie audio.

## Syntaxe

```
outrg kstart, aout1 [,aout2, aout3, ..., aoutN]
```

## Exécution

*kstart* - le numéro du premier canal du périphérique de sortie où écrire (les numéros des canaux commencent à 1, qui est le premier canal).

*aout1*, *aout2*, ... *aoutN* - les arguments contenant les données audio à sortir sur les canaux correspondants.

*outrg* permet la sortie vers un ensemble de canaux contigus du périphérique de sortie audio. *kstart* indique le premier canal où écrire (le canal 1 étant le premier canal). Il faut s'assurer que le nombre obtenu en ajoutant à *kstart* le nombre de canaux à écrire - 1 est  $\leq nchnls$ .

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# outs

outs — Ecrit des données audio stéréo vers un périphérique externe ou un flot.

## Description

Ecrit des données audio stéréo vers un périphérique externe ou un flot.

## Syntaxe

```
outs asig1, asig2
```

## Exécution

Envoie des échantillons stéréo dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Voir Aussi

*out, outh, outh, outq, outq1, outq2, outq3, outq4, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# outs1

outs1 — Ecrit des échantillons vers le canal stéréo n°1 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons vers le canal stéréo n°1 d'un périphérique externe ou d'un flot.

## Syntaxe

`outs1 asig`

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadra n°3, etc.

## Voir Aussi

*out, outh, outh, outq, outq1, outq2, outq3, outq4, outs, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# outs2

outs2 — Ecrit des échantillons vers le canal stéréo n°2 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons vers le canal stéréo n°2 d'un périphérique externe ou d'un flot.

## Syntaxe

`outs2 asig`

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadra n°3, etc.

## Voir Aussi

*out, outh, outh, outq, outq1, outq2, outq3, outq4, outs, outs1, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997



# outvalue

outvalue — Envoie un signal de taux-k ou une chaîne de caractères vers un canal défini par l'utilisateur.

## Description

Envoie un signal de taux-k ou une chaîne de caractères vers un canal défini par l'utilisateur.

## Syntaxe

```
outvalue "channel name", kvalue
```

```
outvalue "channel name", "string"
```

## Exécution

*"channel name"* -- Un entier ou une chaîne de caractères (entre guillemets) représentant le canal.

*kvalue* -- La valeur de taux-k envoyée vers le canal.

*string* -- La constante ou la variable chaîne de caractères envoyée vers le canal.

## Voir Aussi

*invalue*

## Crédits

Auteur : Matt Ingalls

# outx

outx — Ecrit des données audio sur 16 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 16 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outx asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, \  
      asig9, asig10, asig11, asig12, asig13, asig14, asig15, asig16
```

## Exécution

*outx* sort 16 canaux d'audio.

## Voir Aussi

*out32, outc, outch, outz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# outz

outz — Ecrit des données audio multi-canaux depuis un tableau ZAK vers un périphérique externe ou un flot.

## Description

Ecrit des données audio multi-canaux depuis un tableau ZAK vers un périphérique externe ou un flot.

## Syntaxe

```
outz ksig1
```

## Exécution

*outz* envoie en sortie *nchnls* de données audio d'un tableau ZAK.

## Voir Aussi

*out32*, *outc*, *outch*, *outx*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.06 de Csound

# p

p — Montre la valeur contenu dans un p-champ donné.

## Description

Montre la valeur contenu dans un p-champ donné.

## Syntaxe

**p**(x)

Cette fonction tourne au taux-i et au taux-k.

## Initialisation

x -- le numéro du p-champ.

## Exécution

La valeur retournée par la fonction *p* est la valeur contenue dans un p-champ.

## Exemples

Voici un exemple de l'opcode p. Il utilise le fichier *p.csd* [examples/p.csd].

### Exemple 349. Exemple de l'opcode p.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o p.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the value in the fourth p-field, p4.
i1 = p(4)

print i1
endin

</CsInstruments>
```

```
<CsScore>
; p4 = value to be printed.
; Play Instrument #1 for one second, p4 = 50.375.
i 1 0 1 50.375
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celle-ci :

```
instr 1:  i1 = 50.375
```

## Crédits

Exemple écrit par Kevin Conder.

# p5gconnect

p5gconnect — Reads data from a P5 Glove controller.

## Description

Opens and at control-rate polls a P5 Glove controller.

## Syntax

p5gconnect

## Initialization

The opcode locates a P5 Glove attached to the computer by USB, and starts a listener thread to poll the device.

## Performance

Every control cycle the glove is polled for its position, and finger and button states. These values are read by the *p5gdata* opcode.

## Example

Here is an example of the p5g opcodes. It uses the file *p5g.csd* [examples/p5g.csd].

### Exemple 350. Example of the p5g opcodes.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
--rtaudio=alsa -o dac:hw:0
</CsOptions>
<CsInstruments>
nchnls = 1
ksmps = 1000

#define P5G_BUTTONS      #0#
#define P5G_BUTTON_A    #1#
#define P5G_BUTTON_B    #2#
#define P5G_BUTTON_C    #4#
#define P5G_JUSTPUSH     #8#
#define P5G_JUSTPU_A     #9#
#define P5G_JUSTPU_B    #10#
#define P5G_JUSTPU_C    #12#
#define P5G_RELEASED     #16#
#define P5G_RELSED_A     #17#
#define P5G_RELSED_B     #18#
#define P5G_RELSED_C     #20#
#define P5G_FINGER_INDEX #32#
#define P5G_FINGER_MIDDLE #33#
#define P5G_FINGER_RING  #34#
#define P5G_FINGER_PINKY #35#
#define P5G_FINGER_THUMB #36#
#define P5G_DELTA_X      #37#
```

```

#define P5G_DELTA_Y      #38#
#define P5G_DELTA_Z      #39#
#define P5G_DELTA_XR     #40#
#define P5G_DELTA_YR     #41#
#define P5G_DELTA_ZR     #42#
#define P5G_ANGLE        #43#

gka  init 0
gkp  init 0

instr 1
    p5gconnect
    ka  p5gdata  $P5G_JUSTPU_A.
    kc  p5gdata  $P5G_BUTTON_C.
; If the A button is just pressed then activate a note
    if (ka==0)    goto ee
    event        "i", 2, 0, 2

ee:
    gka p5gdata  $P5G_DELTA_X.
    gkp p5gdata  $P5G_DELTA_Y.
    printk2 gka
    printk2 gkp
    if (kc==0)    goto ff
    printks "turning off (%d)\n", 0, kc
    turnoff
ff:
endin

instr 2
    a1 oscil ampdbs(gkp), 440+100*gka, 1
;; a1 oscil 10000, 440, 1
    out a1
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1
i1 0 300

</CsScore>
</CsoundSynthesizer>

```

## See Also

*p5gdata*,

## Credits

Author: John fitch  
 Codemist Ltd  
 2009

New in version 5.12

# p5gdata

p5gdata — Reads data fields from an external P5 Glove.

## Description

Reads data fields from a P5 Glove controller.

## Syntax

```
kres p5gdata kcontrol
```

## Initialization

This opcode must be used in conjunction with a running *p5gconnect* opcode.

## Performance

*kcontrol* -- the code for which control to read

On each access a particular data item of the P5 glove is read. The currently implemented controls are given below, together with the macro name defined in the file *p5g\_mac*:

0 (P5G\_BUTTONS): returns a bit pattern for all buttons that were pressed.

1 (P5G\_BUTTON\_A): returns 1 if the button has just been pressed, or 0 otherwise.

2 (P5G\_BUTTON\_B): as above.

4 (P5G\_BUTTON\_C): as above.

8 (P5G\_JUSTPUSH): returns a bit pattern for all buttons that have just been pressed.

9 (P5G\_JUSTPU\_A): returns 1 if the A button has just been pressed.

10 (P5G\_JUSTPU\_B): as above.

12 (P5G\_JUSTPU\_C): as above.

16 (P5G\_RELEASED): returns a bit pattern for all buttons that have just been released.

17 (P5G\_RELSED\_A): returns 1 if the A button has just been released.

18 (P5G\_RELSED\_B): as above.

20 (P5G\_RELSED\_C): as above.

32 (P5G\_FINGER\_INDEX): returns the clench value of the index finger.

33 (P5G\_FINGER\_MIDDLE): as above.

34 (P5G\_FINGER\_RING): as above.

35 (P5G\_FINGER\_PINKY): as above with little finger.



36 (P5G\_FINGER\_THUMB): as above.

37 (P5G\_DELTA\_X): The X position of the glove.

38 (P5G\_DELTA\_Y): The Y position of the glove.

39 (P5G\_DELTA\_Z): The Z position of the glove.

40 (P5G\_DELTA\_XR): The X axis change (angle).

41 (P5G\_DELTA\_YR): as above.

42 (P5G\_DELTA\_ZR): as above.

43 (P5G\_ANGLES): The general angle

## Examples

See the example for *p5gconnect*.

## See Also

*p5gconnect*,

## Credits

Author: John ffitch  
Codemist Ltd  
2009

New in version 5.12

# pan

pan — Distribute an audio signal amongst four channels.

## Description

Distribute an audio signal amongst four channels with localization control.

## Syntax

`a1, a2, a3, a4 pan asig, kx, ky, ifn [, imode] [, ioffset]`

## Initialization

*ifn* -- function table number of a stored pattern describing the amplitude growth in a speaker channel as sound moves towards it from an adjacent speaker. Requires extended guard-point.

*imode* (optional) -- mode of the *kx*, *ky* position values. 0 signifies raw index mode, 1 means the inputs are normalized (0 - 1). The default value is 0.

*ioffset* (optional) -- offset indicator for *kx*, *ky*. 0 infers the origin to be at channel 3 (left rear); 1 requests an axis shift to the quadraphonic center. The default value is 0.

## Performance

*pan* takes an input signal *asig* and distributes it amongst four outputs (essentially quad speakers) according to the controls *kx* and *ky*. For normalized input (mode=1) and no offset, the four output locations are in order: left-front at (0,1), right-front at (1,1), left-rear at the origin (0,0), and right-rear at (1,0). In the notation (*kx*, *ky*), the coordinates *kx* and *ky*, each ranging 0 - 1, thus control the 'rightness' and 'forwardness' of a sound location.

Movement between speakers is by amplitude variation, controlled by the stored function table *ifn*. As *kx* goes from 0 to 1, the strength of the right-hand signals will grow from the left-most table value to the right-most, while that of the left-hand signals will progress from the right-most table value to the left-most. For a simple linear pan, the table might contain the linear function 0 - 1. A more correct pan that maintains constant power would be obtained by storing the first quadrant of a sinusoid. Since pan will scale and truncate *kx* and *ky* in simple table lookup, a medium-large table (say 8193) should be used.

*kx*, *ky* values are not restricted to 0 - 1. A circular motion passing through all four speakers (inscribed) would have a diameter of root 2, and might be defined by a circle of radius  $R = \text{root } 1/2$  with center at (.5,.5). *kx*, *ky* would then come from  $R\cos(\text{angle})$ ,  $R\sin(\text{angle})$ , with an implicit origin at (.5,.5) (i.e. *ioffset* = 1). Unscaled raw values operate similarly. Sounds can thus be located anywhere in the polar or Cartesian plane; points lying outside the speaker square are projected correctly onto the square's perimeter as for a listener at the center.

## Examples

```
instr      1
  k1      phasor    1/p3          ; fraction of circle
  k2      tablei    k1, 1, 1      ; sin of angle (sinusoid in f1)
  k3      tablei    k1, 1, 1, .25, 1 ; cos of angle (sin offset 1/4 circle)
```

```

a1      oscili      10000,440, 1      ; audio signal..
a1,a2,a3,a4 pan      a1, k2/2, k3/2, 2, 1, 1 ; sent in a circle (f2=1st quad sin)
                                outq a1, a2, a3, a4
endin
```

# pan2

pan2 — Distribute an audio signal across two channels.

## Description

Distribute an audio signal across two channels with a choice of methods.

## Syntax

```
a1, a2 pan2 asig, xp [, imode]
```

## Initialization

*imode* (optional) -- mode of the stereo positioning algorithm. 0 signifies equal power (harmonic) panning, 1 means the square root method, 2 means simple linear and 3 means an alternative equal-power pan (based on an UDO). The default value is 0.

## Performance

*pan2* takes an input signal *asig* and distributes it across two outputs (essentially stereo speakers) according to the control *xp* which can be k- or a-rate. A zero value for *xp* indicates hard left, and a 1 is hard right.

## Examples

```
instr      1
  kline line    0, p3, 1      ; straight line
  ain oscili 10000,440, 1    ; audio signal..
  a1,a2 pan2   ain, kline    ; sent across image

endin      outs   a1, a2
```

## Credits

Author: John ffitch  
University of Bath, Codemist Ltd.  
Bath, UK  
September 2007

New in version 5.07

## pareq

pareq — Implementation of Zoelzer's parametric equalizer filters.

### Description

Implementation of Zoelzer's parametric equalizer filters, with some modifications by the author.

The formula for the low shelf filter is:

$$\begin{aligned}\omega &= 2\pi f / sr \\ K &= \tan(\omega/2) \\ b0 &= 1 + \sqrt{2V}K + V K^2 \\ b1 &= 2(V K^2 - 1) \\ b2 &= 1 - \sqrt{2V}K + V K^2 \\ a0 &= 1 + K/Q + K^2 \\ a1 &= 2(K^2 - 1) \\ a2 &= 1 - K/Q + K^2\end{aligned}$$

The formula for the high shelf filter is:

$$\begin{aligned}\omega &= 2\pi f / sr \\ K &= \tan((\pi - \omega)/2) \\ b0 &= 1 + \sqrt{2V}K + V K^2 \\ b1 &= -2(V K^2 - 1) \\ b2 &= 1 - \sqrt{2V}K + V K^2 \\ a0 &= 1 + K/Q + K^2 \\ a1 &= -2(K^2 - 1) \\ a2 &= 1 - K/Q + K^2\end{aligned}$$

The formula for the peaking filter is:

$$\begin{aligned}\omega &= 2\pi f / sr \\ K &= \tan(\omega/2) \\ b0 &= 1 + V K/2 + K^2 \\ b1 &= 2(K^2 - 1) \\ b2 &= 1 - V K/2 + K^2 \\ a0 &= 1 + K/Q + K^2 \\ a1 &= 2(K^2 - 1) \\ a2 &= 1 - K/Q + K^2\end{aligned}$$

## Syntax

```
ares pareq asig, kc, kv, kq [, imode] [, iskip]
```

## Initialization

*imode* (optional, default: 0) -- operating mode

- 0 = Peaking
- 1 = Low Shelving
- 2 = High Shelving

*iskip* (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

## Performance

*kc* -- center frequency in peaking mode, corner frequency in shelving mode.

*kv* -- amount of boost or cut. A value less than 1 is a cut. A value greater than 1 is a boost. A value of 1 is a flat response.

*kq* -- Q of the filter (sqrt(.5) is no resonance)

*asig* -- the incoming signal

## Examples

Here is an example of the `pareq` opcode. It uses the file `pareq.csd` [examples/pareq.csd].

### Exemple 351. Example of the `pareq` opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pareq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 15
  ifc      =      p4      ; Center / Shelf
  kq       =      p5      ; Quality factor sqrt(.5) is no resonance
  kv       =      ampdb(p6) ; Volume Boost/Cut
  imode    =      p7      ; Mode 0=Peaking EQ, 1=Low Shelf, 2=High Shelf
```

```

kfc      linseg  ifc*2, p3, ifc/2
asig      rand   5000                ; Random number source for testing
aout      pareq  asig, kfc, kv, kq, imode ; Parmetric equalization
          outs   aout, aout           ; Output the results
endin

</CsInstruments>
<CsScore>

; SCORE:
;   Sta  Dur  Fcenter  Q          Boost/Cut(dB)  Mode
i15 0    1    10000   .2          12             1
i15 +    .    5000   .2          12             1
i15 .    .    1000   .707       -12             2
i15 .    .    5000   .1         -12             0
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Hans Mikelson  
December 1998

New in Csound version 3.50

# partials

partials — Partial track spectral analysis.

## Description

The `partials` opcode takes two input PV streaming signals containing `AMP_FREQ` and `AMP_PHASE` signals (as generated for instance by `pvsifd` or in the first case, by `pvsanal`) and performs partial track analysis, as described in Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc.of ICMC05, Barcelona. It generates a `TRACKS` PV streaming signal, containing amplitude, frequency, phase and track ID for each output track. This type of signal will contain a variable number of output tracks, up to the total number of analysis bins contained in the inputs ( $\text{fftsize}/2 + 1$  bins). The second input (`AMP_PHASE`) is optional, as it can take the same signal as the first input. In this case, however, all phase information will be `NULL` and resynthesis using phase information cannot be performed.

## Syntax

```
ftrks partials ffr, fphs, kthresh, kminpts, kmaxgap, imaxtracks
```

## Performance

*ftrks* -- output pv stream in `TRACKS` format

*ffr* -- input pv stream in `AMP_FREQ` format

*fphs* -- input pv stream in `AMP_PHASE` format

*kthresh* -- analysis threshold. Tracks below  $\text{kthresh} * \text{max\_magnitude}$  will be discarded ( $1 > \text{kthresh} \geq 0$ ).

*kminpoints* -- minimum number of time points for a detected peak to make a track (1 is the minimum). Since this opcode works with streaming signals, larger numbers will increase the delay between input and output, as we have to wait for the required minimum number of points.

*kmaxgap* -- maximum gap between time-points for track continuation ( $> 0$ ). Tracks that have no continuation after *kmaxgap* will be discarded.

*imaxtracks* -- maximum number of analysis tracks (number of bins  $\geq$  *imaxtracks*)

## Examples

### Exemple 352. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
    aout resyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```



The example above shows partial tracking of an ifd-analysis signal and cubic-phase additive resynthesis with pitch shifting.

## Credits

Author: Victor Lazzarini  
June 2005

New plugin in version 5

November 2004.

# partikkel

*partikkel* — Synthétiseur granulaire avec un contrôle "par grain" grâce à ses nombreux paramètres. Il a une entrée sync pour synchroniser son horloge interne de distribution des grains avec une horloge externe.

## Description

*partikkel* a été conçu après la lecture du livre de Curtis Road "Microsound", et le but était de créer un opcode capable de réaliser toutes les variétés temporelles de synthèse granulaire décrites dans ce livre. L'idée étant que la plupart des techniques ne diffèrent que par les valeurs des paramètres, et que si l'on a un opcode unique qui peut produire toutes les variétés de synthèse granulaire, l'interpolation entre ces techniques devient possible. La synthèse granulaire est parfois appelée synthèse par particules et il m'a semblé approprié de nommer l'opcode *partikkel* afin de le distinguer des autres opcodes granulaires.

Certains des paramètres d'entrée de *partikkel* sont des numéros de table, pointant sur des tables dans lesquelles sont mémorisées des valeurs pour les changements de paramètre "par grain". *partikkel* peut utiliser une période d'une forme d'onde ou des formes d'onde complexes (par exemple un son échantillonné) comme source de forme d'onde pour les grains. Chaque grain est constitué du mélange de 4 formes d'onde source. On peut accorder séparément la fréquence de base de chacune des 4 formes d'onde source. La modulation de fréquence à l'intérieur de chaque grain est activée via une entrée audio auxiliaire (*awavfm*). La synthèse par trainlet (un trainlet est un bref train d'impulsions) est possible, et les trainlets peuvent être mélangés avec des grains basés sur des tables d'onde. On peut utiliser jusqu'à 8 sorties audio séparées.

## Syntaxe

```
a1 [, a2, a3, a4, a5, a6, a7, a8] partikkel agrainfreq, \  
kdistribution, idisttab, async, kenv2amt, ienv2tab, ienv_attack, \  
ienv_decay, ksustain_amount, ka_d_ratio, kduration, kamp, igainmask, \  
kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \  
ifmampstab, kfmenv, icosine, ktraincps, knumpartials, kchroma, \  
ichannelmask, krandommask, kwaveform1, kwaveform2, kwaveform3, \  
kwaveform4, iwaveamptab, asamplepos1, asamplepos2, asamplepos3, \  
asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains \  
[, iopcode_id]
```

## Initialisation

*idisttab* -- numéro d'une table de fonction, distribution des déplacements aléatoires du grain dans le temps. Les valeurs de la table sont interprétées comme la "quantité de déplacement" pondérée par 1/(rythme des grains). Cela signifie qu'une valeur de 0,5 dans la table déplacera un grain de la moitié de la période du rythme des grains. Les valeurs de la table sont lues aléatoirement, et pondérées par *kdistribution*. Pour obtenir des résultats stochastiques réalistes, il vaut mieux ne pas utiliser une taille de table trop petite, car cela limite le nombre des valeurs de déplacement possibles. On peut l'exploiter à d'autres fins, par exemple utiliser des valeurs de déplacement quantifiées pour travailler avec des décalages contrôlés à partir de la période du rythme des grains. Si *kdistribution* est négatif, les valeurs de la table seront lues séquentiellement. On peut sélectionner une table par défaut au moyen du numéro de table -1, pour lequel *idisttab* fournit une distribution nulle (pas de déplacement).

*ienv\_attack* -- numéro d'une table de fonction, forme de l'attaque du grain. Il faut un point de garde d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv\_attack* fournit une fenêtre rectangulaire (pas d'enveloppe).

*ienv\_decay* -- numéro d'une table de fonction, forme de la chute du grain. Il faut un point de garde

d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv\_decay* fournit une fenêtre rectangulaire (pas d'enveloppe).

*ienv2tab* -- numéro d'une table de fonction, enveloppe additionnelle appliquée au grain après les enveloppes d'attaque et de chute. On peut l'utiliser par exemple pour la synthèse par formant fof. Il faut un point de garde d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv2tab* fournit une fenêtre rectangulaire (pas d'enveloppe).

*icosine* -- numéro d'une table de fonction, devant contenir un cosinus, utilisée pour les trainlets. La table doit avoir une taille d'au moins 2048 pour obtenir des trainlets de bonne qualité.

*igainmasks* -- numéro d'une table de fonction, gain par grain. La suite des valeurs dans la table a la signification suivante : la valeur d'indice 0 est le point de début d'une boucle de lecture des valeurs, la valeur d'indice 1 étant le point de fin de cette boucle. Les entrées aux autres indices contiennent les valeurs de gain (normalement dans l'intervalle 0 - 1, mais d'autres valeurs sont permises, les valeurs négatives inversant la phase de la forme d'onde du grain) pour une suite de grains ; ces valeurs sont lues au rythme des grains, ce qui permet une correspondance exacte de "gain par grain". Les points du début et de la fin de la boucle sont basés sur zéro avec une origine à l'indice 2, par exemple une valeur de début de boucle de 0 et une valeur de fin de boucle de 3 provoqueront la lecture des valeurs d'indice 2, 3, 4, 5 dans une boucle évoluant au rythme des grains. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *igainmasks* désactive le masquage du gain (tous les grains reçoivent un masque de gain égal à 1).

*ichannelmasks* -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. L'intervalle des valeurs va de 0 à N, où N est le nombre de canaux de sortie moins 1. Une valeur de zéro enverra le grain sur la sortie audio 1 de l'opcode. On peut utiliser des valeurs non entières, par exemple 3,5 répartira le grain également entre les sorties 4 et 5. L'utilisateur doit éviter les dépassements de niveau, aucun test n'étant effectué. L'opcode plantera si des valeurs dépassent le niveau maximal. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ichannelmasks* désactive le masquage des canaux (tous les grains reçoivent un masque de canal de 0 et sont envoyés sur la sortie audio 1 de *partikkel*).

*iwavfreqstarttab* -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. Multiplicateur de la fréquence de départ de chaque grain. La hauteur glissera de la fréquence de départ jusqu'à la fréquence de fin suivant une droite ou une courbe fixée par *ksweepshape*. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwavfreqstarttab* fournit un multiplicateur de 1, désactivant toute modification de la fréquence de départ.

*iwavfreqendtab* -- numéro d'une table de fonction, voir *iwavfreqstarttab*. Multiplicateur de la fréquence de fin de chaque grain. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwavfreqendtab* fournit un multiplicateur de 1, désactivant toute modification de la fréquence de fin.

*ifmamptab* -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. Indice de modulation de fréquence par grain. Le signal *awavfm* sera multiplié par les valeurs lues dans cette table. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ifmamptab* fournit 1 comme indice de modulation, activant la modulation de fréquence pour tous les grains.

*iwaveamptab* -- numéro d'une table de fonction, les indices sont parcourus de la même manière que pour *igainmasks*. La valeur d'indice 0 sert de point de début de boucle et la valeur d'indice 1 de point de fin. Les autres indices sont lus par groupes de 5, dans lesquels chaque valeur représente une valeur de gain pour chacune des 4 formes d'onde source, et la cinquième valeur représente l'amplitude de trainlet. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwaveamptab* fournit un mélange égal des 4 formes d'onde source (chacune avec une amplitude de 0,5) et une amplitude de trainlet nulle.

Le calcul des trainlets étant très gourmand en ressources CPU, on peut éviter la plupart des calculs de

trainlet en fixant *ktrainamp* à zéro. Les trainlets sont normalisés au niveau de crête (*ktrainamp*), en compensation des variations d'amplitude causées par les variations de *kpartials* et de *kchroma*.

*imax\_grains* -- nombre maximum de grains par k-période. Une grande valeur ne devrait pas affecter l'exécution, le dépassement de cette valeur conduira à l'effacement des grains les "plus anciens".

*iopcode\_id* -- identificateur de l'opcode, liant une instance de *partikkel* à une instance de *partikkelsync*, laquelle fournira en sortie des impulsions de déclenchement synchronisées pour le distributeur de grains de *partikkel*. La valeur par défaut est zéro, ce qui signifie aucune connexion à une instance de *partikkelsync*.

## Exécution

*xgrainfreq* -- nombre de grains par seconde. On peut spécifier une valeur nulle, ce qui déléguera la distribution des grains à l'entrée de synchronisation.

*async* -- entrée de synchronisation. Les valeurs entrées sont ajoutées à la phase de l'horloge interne du distributeur de grains, ce qui permet une synchronisation de tempo avec une horloge externe. Comme c'est un signal de taux-a, les entrées sont généralement des impulsions de longueur 1/sr. A l'aide de telles impulsions on peut "faire bouger" la phase interne en avant ou en arrière, ce qui permet une synchronisation plus ou moins forte. Des valeurs d'entrée négatives décrémentent la phase interne, tandis que des valeurs positives dans l'intervalle de 0 à 1 incrémentent la phase interne. Une valeur d'entrée de 1 forcera toujours *partikkel* à générer un grain. Si la valeur reste à 1, l'horloge interne du distributeur de grain marquera une pause mais tous les grains en cours d'exécution continueront jusqu'à leur terme.

*kdistribution* -- distribution périodique ou stochastique des grains, 0 = périodique. Le déplacement stochastique de grain est de l'ordre de *kdistribution/grainrate* secondes. Le profil de la distribution stochastique (distribution aléatoire) peut être fixé dans la table *idisttab*. Si *kdistribution* est négatif, le résultat est un déplacement temporel déterministe comme décrit par *idisttab* (lecture séquentielle des valeurs de déplacement). Le déplacement de grain maximum est limité dans tous les cas à 10 secondes, et un grain conservera les valeurs (durée, hauteur, etc) reçues lors de sa première génération (avant le déplacement temporel). Comme le déplacement de grain dépend du taux de grains, ce déplacement est indéfini pour un taux de grain de 0Hz et *kdistribution* est complètement désactivé dans ce cas.

*kenv2amt* -- dosage de l'enveloppe secondaire dans l'enveloppe de chaque grain. L'intervalle va de 0 à 1, où 0 signifie pas d'enveloppe secondaire (fenêtre rectangulaire), 0,5 provoquera une interpolation entre une fenêtre rectangulaire et la forme fixée par *ienv2tab*.

*ksustain\_amount* -- durée d'entretien exprimée comme une fraction de la durée du grain. C-à-d la proportion entre le temps d'enveloppe (attaque + chute) et le temps d'entretien. Le niveau d'entretien est celui de la dernière valeur de la ftable *ienv\_attack*.

*ka\_d\_ratio* -- proportion entre le temps d'attaque et le temps de chute. Par exemple, avec *ksustain\_amount* à 0,5 et *ka\_d\_ratio* à 0,5, l'enveloppe d'attaque de chaque grain prendra 25% de la durée du grain, l'amplitude maximale (entretien) sera tenue pendant 50% de la durée du grain, et l'enveloppe de chute prendra les 25% restants de la durée du grain.

*kduration* -- durée du grain en millisecondes.

*kamp* -- facteur de pondération de l'amplitude en sortie de l'opcode. Multiplié par l'amplitude de chaque grain lue à partir de *igainmasks*. La lecture de la forme d'onde source dans les grains peut consommer un nombre significatif de cycles CPU, spécialement si la durée de grain est longue, de nombreux grains se chevauchant. Si *kamp* vaut zéro la lecture de la forme d'onde dans les grains n'aura pas lieu (et aucun son ne sera évidemment généré). On peut utiliser cette possibilité comme une méthode de court-circuit "logiciel" si l'on veut garder l'opcode actif mais silencieux pendant un certain temps.

*kwavfreq* -- facteur de transposition. Multiplié par les valeurs de transposition de départ et de fin lues à partir de *iwavfreqstarttab* et de *iwavfreqendtab*.

*ksweepshape* -- forme de la progression de la transposition, contrôle la courbure de la progression de la transposition. Dans l'intervalle de 0 à 1. Avec les valeurs faibles, la transposition sera maintenue plus longtemps près de la valeur de départ puis ira rapidement vers la valeur de fin, tandis qu'avec les valeurs fortes la transposition ira tout de suite rapidement vers la valeur de fin. Une valeur de 0,5 donnera une progression linéaire. La valeur 0 supprimera la progression et ne gardera que la fréquence de départ, tandis que la valeur 1 supprimera la progression et ne gardera que la fréquence de fin. Le générateur de la progression peut être légèrement imprécis lorsqu'il atteint la fréquence finale si l'on utilise une courbe raide avec des grains très longs.

*awavfm* -- entrée audio pour la modulation de fréquence du grain.

*kfmenv* -- numéro d'une table de fonction, enveloppe du signal modulateur de la modulation de fréquence provoquant un changement de l'indice de modulation sur toute la durée du grain.

*ktraincps* -- fréquence fondamentale des trainlets.

*knumpartials* -- nombre de partiels dans les trainlets.

*kchroma* -- couleur spectrale des trainlets. Une valeur de 1 donne une amplitude égale à chaque partiel, des valeurs plus grandes réduiront l'amplitude des partiels inférieurs tout en renforçant l'amplitude des partiels supérieurs.

*krandommask* -- masquage aléatoire (escamotage) de grains individuels. Dans l'intervalle de 0 à 1, où la valeur 0 signifie pas de masquage (tous les grains sont joués), et la valeur 1 escamote tous les grains.

*kwaveform1* -- numéro de la table pour la forme d'onde source 1.

*kwaveform2* -- numéro de la table pour la forme d'onde source 2.

*kwaveform3* -- numéro de la table pour la forme d'onde source 3.

*kwaveform4* -- numéro de la table pour la forme d'onde source 4.

*asamplepos1* -- position de départ pour la lecture de la forme d'onde source 1.

*asamplepos2* -- position de départ pour la lecture de la forme d'onde source 2.

*asamplepos3* -- position de départ pour la lecture de la forme d'onde source 3.

*asamplepos4* -- position de départ pour la lecture de la forme d'onde source 4.

*kwavekey1* -- hauteur originale de la forme d'onde source 1. On peut l'utiliser pour transposer chaque forme d'onde source indépendamment.

*kwavekey2* -- comme *kwavekey1*, mais pour la forme d'onde source 2.

*kwavekey3* -- comme *kwavekey1*, mais pour la forme d'onde source 3.

*kwavekey4* -- comme *kwavekey1*, mais pour la forme d'onde source 4.

## Exemples

Voici un exemple de l'opcode *partikkel*. Il utilise le fichier *PartikkelExample1.csd* [exemples/PartikkelExample1.csd].

### Exemple 353. Exemple de l'opcode *partikkel*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac          ;;RT audio
; For Non-realtime ouput leave only the line below:
; -o partikkel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 20
nchnls = 2

; Example by Joachim Heintz and Oeyvind Brandtsegg 2008

giCosine ftgen 0, 0, 8193, 9, 1, 1, 90          ; cosine
giDisttab ftgen 0, 0, 32768, 7, 0, 32768, 1 ; for kdistribution
giFile    ftgen 0, 0, 0, 1, "fox.wav", 0, 0, 0 ; soundfile for source waveform
giWin     ftgen 0, 0, 4096, 20, 9, 1          ; grain envelope
giPan     ftgen 0, 0, 32768, -21, 1          ; for panning (random values between 0 and 1)

; *****
; partikkel example, processing of soundfile
; uses the file "fox.wav"
; *****
instr 1

/*score parameters*/
ispeed          = p4          ; 1 = original speed
igrainrate      = p5          ; grain rate
igrainsize      = p6          ; grain size in ms
icent           = p7          ; transposition in cent
iposrand        = p8          ; time position randomness (offset) of the pointer in ms
icentrand       = p9          ; transposition randomness in cents
ipan            = p10         ; panning narrow (0) to wide (1)
idist           = p11         ; grain distribution (0=periodic, 1=scattered)

/*get length of source wave file, needed for both transposition and time pointer*/
ifilen          tablen giFile
ifildur         = ifilen / sr

/*sync input (disabled)*/
async          = 0

/*grain envelope*/
kenv2amt        = 1          ; use only secondary envelope
ienv2tab        = giWin      ; grain (secondary) envelope
ienv_attack     = -1          ; default attack envelope (flat)
ienv_decay      = -1          ; default decay envelope (flat)
ksustain_amount = 0.5         ; no meaning in this case (use only secondary envelope, ienv2tab)
ka_d_ratio      = 0.5         ; no meaning in this case (use only secondary envelope, ienv2tab)

/*amplitude*/
kamp            = 0.4*0dbfs ; grain amplitude
igainmask      = -1          ; (default) no gain masking

/*transposition*/
kcentrand       rand icentrand ; random transposition
iorig           = 1 / ifildur ; original pitch
kwavfreq        = iorig * cent(icent + kcentrand)

/*other pitch related (disabled)*/
ksweepshape     = 0          ; no frequency sweep
iwavfreqstarttab = -1        ; default frequency sweep start
iwavfreqendtab  = -1        ; default frequency sweep end
awavfm          = 0          ; no FM input
ifmamptab       = -1        ; default FM scaling (=1)
kfmenv          = -1        ; default FM envelope (flat)

/*trainlet related (disabled)*/
icosine         = giCosine ; cosine ftable
kTrainCps       = igrainrate ; set trainlet cps equal to grain rate for single-cycle trainlet in each
knumpartials    = 1          ; number of partials in trainlet
```

```

kchroma                = 1          ; balance of partials in trainlet

/*panning, using channel masks*/
imid                   = .5; center
ileftmost              = imid - ipan/2
irightmost             = imid + ipan/2
giPanthis              ftgen 0, 0, 32768, -24, giPan, ileftmost, irightmost ; rescales giPan according to ip
                        tableiw      0, 0, giPanthis          ; change index 0 ...
                        tableiw      32766, 1, giPanthis       ; ... and 1 for ichannelmas

ichannelmasks          = giPanthis ; ftable for panning

/*random gain masking (disabled)*/
krandommask            = 0

/*source waveforms*/
kwaveform1             = giFile ; source waveform
kwaveform2             = giFile ; all 4 sources are the same
kwaveform3             = giFile
kwaveform4             = giFile
iwaveamptab            = -1        ; (default) equal mix of source waveforms and no amplitude for trainlet

/*time pointer*/
afilposphas            phasor ispeed / ifildur
/*generate random deviation of the time pointer*/
iposrandsec            = iposrand / 1000 ; ms -> sec
iposrand               = iposrandsec / ifildur ; phase values (0-1)
krndpos               linrand iposrand ; random offset in phase values
/*add random deviation to the time pointer*/
asamplepos1            = afilposphas + krndpos; resulting phase values (0-1)
asamplepos2            = asamplepos1
asamplepos3            = asamplepos1
asamplepos4            = asamplepos1

/*original key for each source waveform*/
kwavekey1              = 1
kwavekey2              = kwavekey1
kwavekey3              = kwavekey1
kwavekey4              = kwavekey1

/* maximum number of grains per k-period*/
imax_grains            = 100

aL, aR                 partikkel igrainrate, idist, giDisttab, async, kenv2amt, ienv2tab, \
                        ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, igrainsize, kamp, igainmasks, \
                        kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
                        ifmamptab, kfmenv, icosine, kTrainCps, knumpartials, \
                        kchroma, ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
                        iwaveamptab, asamplepos1, asamplepos2, asamplepos3, asamplepos4, \
                        kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains

                        outs          aL, aR

endin

</CsInstruments>
<CsScore>
;il st dur speed grate gsize cent posrnd cntrnd pan dist
i1 0 2.757 1 200 15 0 0 0 0 0
s
i1 0 2.757 1 200 15 400 0 0 0 0
s
i1 0 2.757 1 15 450 400 0 0 0 0
s
i1 0 2.757 1 15 450 400 0 0 0 0.4
s
i1 0 2.757 1 200 15 0 400 0 0 1
s
i1 0 5.514 .5 200 20 0 0 600 .5 1
s
i1 0 11.028 .25 200 15 0 1000 400 1 1

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode partikkel. Il utilise le fichier *partikkel\_softsync.csd* [examples/partikkel\_softsync.csd].

### Exemple 354. Exemple avec une synchronisation légère de deux générateurs partikkel.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac          ;;RT audio
; For Non-realtime ouput leave only the line below:
; -o partikkel_softsync.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 20
nchnls = 2

; Example by Oeyvind Brandtsegg 2007, revised 2008

giSine          ftgen 0, 0, 65537, 10, 1
giCosine ftgen 0, 0, 8193, 9, 1, 1, 90
giSigmoidRise ftgen 0, 0, 8193, 19, 0.5, 1, 270, 1 ; rising sigmoid
giSigmoidFall ftgen 0, 0, 8193, 19, 0.5, 1, 90, 1 ; falling sigmoid

; *****
; example of soft synchronization of two partikkel instances
; *****
instr 1

/*score parameters*/
igrainrate = p4 ; grain rate
igrainsize = p5 ; grain size in ms
igrainFreq = p6 ; fundamental frequency of source waveform
iosc2Dev = p7 ; partikkel instance 2 grain rate deviation factor
iMaxSync = p8 ; max soft sync amount (increasing to this value during length of note)

/*overall envelope*/
iattack = 0.001
idecay = 0.2
isustain = 0.7
irelease = 0.2
amp linsegr 0, iattack, 1, idecay, isustain, 1, isustain, irelease, 0

kgrainfreq = igrainrate ; grains per second
kdistribution = 0 ; periodic grain distribution
idisttab = -1 ; (default) flat distribution used ; for grain distribution

async = 0 ; no sync input
kenv2amt = 0 ; no secondary enveloping
ienv2tab = -1 ; default secondary envelope (flat)
ienv_attack = giSigmoidRise ; default attack envelope (flat)
ienv_decay = giSigmoidFall ; default decay envelope (flat)
ksustain_amount = 0.3 ; time (in fraction of grain dur) at ; sustain level for each grain ; balance between attack and decay time

ka_d_ratio = 0.2 ; set grain duration in ms
kduration = igrainsize ; amp
kamp = 0.2*0dbfs ; (default) no gain masking
igainmask = -1 ; fundamental frequency of source waveform
kwaifreq = igrainFreq ; shape of frequency sweep (0=no sweep)
ksweepshape = 0 ; default frequency sweep start ; (value in table = 1, which give ; no frequency modification)
iwaifreqstarttab = -1 ; default frequency sweep end ; (value in table = 1, which give ; no frequency modification)

iwaifreqendtab = -1 ; no FM input
awavfm = 0 ; default FM scaling (=1)
ifmamptab = -1 ; default FM envelope (flat)
kfmenv = -1 ; cosine ftable
icosine = giCosine ; set trainlet cps equal to grain ; rate for single-cycle trainlet in ; each grain
kTrainCps = kgrainfreq ; number of partials in trainlet

knumpartials = 3

```



```

kchroma      = 1          ; balance of partials in trainlet
ichannelmasks = -1       ; (default) no channel masking,
                           ; all grains to output 1
krandommask  = 0          ; no random grain masking
kwaveform1   = giSine     ; source waveforms
kwaveform2   = giSine     ;
kwaveform3   = giSine     ;
kwaveform4   = giSine     ;
iwaveamptab  = -1         ; mix of 4 source waveforms and
                           ; trainlets (set to default)
asamplepos1  = 0          ; phase offset for reading source waveform
asamplepos2  = 0          ;
asamplepos3  = 0          ;
asamplepos4  = 0          ;
kwavekey1    = 1          ; original key for source waveform
kwavekey2    = 1          ;
kwavekey3    = 1          ;
kwavekey4    = 1          ;
imax_grains  = 100        ; max grains per k period
iopcode_id   = 1          ; id of opcode, linking partikkel
                           ; to partikkelsync

a1 partikkel kgrainfreq, kdistribution, idisttab, async, kenv2amt, \
    ienv2tab, ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, \
    kduration, kamp, igainmasks, kwavfreq, ksweepshape, \
    iwavfreqstarttab, iwavfreqendtab, awavfm, ifmamp, kfmenv, \
    icosine, kTrainCps, knumpartials, kchroma, ichannelmasks, \
    krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
    iwaveamptab, asamplepos1, asamplepos2, asamplepos3, asamplepos4, \
    kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains, iopcode_id

async1 partikkelsync iopcode_id ; clock pulse output of the
                                   ; partikkel instance above
ksyncGravity line 0, p3, iMaxSync ; strength of synchronization
aphase2 init 0
asyncPolarity limit (int(aphase2*2)*2)-1, -1, 1
; use the phase of partikkelsync instance 2 to find sync
; polarity for partikkel instance 2.
; If the phase of instance 2 is less than 0.5, we want to
; nudge it down when synchronizing,
; and if the phase is > 0.5 we want to nudge it upwards.
async1 = async1*ksyncGravity*asyncPolarity ; prepare sync signal
                                   ; with polarity and strength

kgrainfreq2 = igrainrate * iosc2Dev ; grains per second for second partikkel instance
iopcode_id2 = 2
a2 partikkel kgrainfreq2, kdistribution, idisttab, async1, kenv2amt, \
    ienv2tab, ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, \
    kduration, kamp, igainmasks, kwavfreq, ksweepshape, \
    iwavfreqstarttab, iwavfreqendtab, awavfm, ifmamp, kfmenv, \
    icosine, kTrainCps, knumpartials, kchroma, ichannelmasks, \
    krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
    iwaveamptab, asamplepos1, asamplepos2, asamplepos3, \
    asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, \
    imax_grains, iopcode_id2

async2, aphase2 partikkelsync iopcode_id2
; clock pulse and phase
; output of the partikkel instance above,
; we will only use the phase

outs a1*amp, a2*amp

endin

</CsInstruments>
<CsScore>

/*score parameters
igrainrate = p4          ; grain rate
igrainsize = p5          ; grain size in ms
igrainfreq = p6          ; frequency of source wave within grain
iosc2Dev = p7            ; partikkel instance 2 grain rate deviation factor
iMaxSync = p8            ; max soft sync amount (increasing to this value during length of note)
*/
;          GrRate GrSize GrFund Osc2Dev MaxSync

i1 0      10 2 20 880 1.3 0.3
s
i1 0      10 5 20 440 0.8 0.3
s

```

```
i1 0      6 55 15 660 1.8 0.45
s
i1 0      6 110 10 440 0.6 0.6
s
i1 0      6 220 3 660 2.6 0.45
s
i1 0      6 220 3 660 2.1 0.45
s
i1 0      6 440 3 660 0.8 0.22
s

e

e
</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*fof, fof2, fog, grain, grain2, grain3, granule, sndwarp, sndwarpst, syncgrain, syncloop, partikkelsync*

## Crédits

Auteur : Thom Johansen  
Auteur : Torgeir Strand Henriksen  
Auteur : Øyvind Brandtsegg  
Avril 2007

Exemples écrits par Joachim Heintz et Øyvind Brandtsegg.

Nouveau dans la version 5.06

# partikkelsync

*partikkelsync* — Produit l'impulsion et la phase de l'horloge du distributeur de grain de *partikkel* pour synchroniser plusieurs instances de l'opcode *partikkel* à la même source d'horloge.

## Description

*partikkelsync* est un opcode dont la tâche est de produire l'impulsion et la phase de l'horloge du distributeur de grain de *partikkel*. On peut utiliser la sortie de *partikkelsync* pour synchroniser d'autres instances de l'opcode *partikkel* à la même horloge.

## Syntaxe

```
async [,aphase] partikkelsync opcode_id
```

## Initialisation

*opcode\_id* -- identificateur de l'opcode, liant une instance de *partikkel* à une instance de *partikkelsync*.

## Exécution

*async* -- signal d'impulsion de déclenchement. Envoie des impulsions de déclenchement synchronisées à l'horloge du distributeur de grain d'un opcode *partikkel*. Une impulsion de déclenchement est générée pour le démarrage de chaque grain dans l'opcode *partikkel* ayant le même *opcode\_id*. Dans une utilisation normale, on enverra ce signal à l'entrée *async* d'un autre opcode *partikkel* pour synchroniser plusieurs instances de *partikkel*.

*aphase* -- phase de l'horloge. Sort un signal de phase linéaire. On peut l'utiliser par exemple pour une synchronisation légère, ou simplement comme un générateur de phase à la *phasor*.

## Voir Aussi

*partikkel*

## Crédits

Auteur : Thom Johansen  
Auteur : Torgeir Strand Henriksen  
Auteur : Øyvind Brandtsegg  
Avril 2007

Nouveau dans la version 5.06

# passign

passign — Affecte un ensemble de p-champs à des variables de taux i.

## Description

Affecte un ensemble de p-champs à des variables de taux i.

## Syntaxe

```
ivar1, ... passign [istart]
```

## Initialisation

L'argument optionnel *istart* donne l'indice du premier p-champ à affecter. La valeur par défaut est 1, ce qui correspond au numéro d'instrument.

Une des variables peut être une variable chaîne de caractères, à laquelle sera affecté dans ce cas le seul paramètre de type chaîne de caractères, s'il y en a un, sinon une erreur.

## Exécution

*passign* transfère les p-champs de l'instrument à des variables de l'instrument, en commençant par celui qui est identifié par l'argument *istart*. Il ne doit pas y avoir plus de variables que de p-champs, mais il peut y en avoir moins.

## Voir Aussi

*assign*,

## Exemple

### Exemple 355. Une variante de toot8.csd qui utilise passign.

Voici un exemple de l'opcode *passign*. Il utilise le fichier *passign.csd* [examples/passign.csd].

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

    instr 8

idur,iamp,iskiptime,iattack,irelease,irvbtime,irvbgain passign 3

kamp      linen      iamp, iattack, idur, irelease
asig      soundin    "fox.wav", iskiptime
arampsig  =           kamp * asig
aeffect   reverb     asig, irvbtime
arvbretrn =           aeffect * irvbgain

    out              arampsig + arvbretrn
```

```
        endin

</CsInstruments>
<CsScore>

;ins strt dur amp skip atk rel rvbt rvbgain
i8 0 2.28 .3 0 .03 .1 1.5 .3
i8 4 1.6 .3 1.6 .1 .1 1.1 .4
i8 5.5 2.28 .3 0 .5 .1 2.1 .2
i8 6.5 2.28 .4 0 .01 .1 1.1 .1
i8 8 2.28 .5 0.1 .01 .1 0.1 .1

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur: John ffitch  
Université de Bath, Codemist Ltd.  
Bath, UK  
Décembre 2009

Nouveau dans la version 5.12

# pcauchy

pcauchy — Générateur de nombres aléatoires de distribution de Cauchy (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution de Cauchy (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

ares **pcauchy** kalpha

ires **pcauchy** kalpha

kres **pcauchy** kalpha

## Exécution

*pcauchy kalpha* -- contrôle l'étalement à partir de zéro (grand *kalpha* = grand étalement). Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode *pcauchy*. Il utilise le fichier *pcauchy.csd* [examples/pcauchy.csd].

### Exemple 356. Exemple de l'opcode *pcauchy*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pcauchy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
```

```
nchnls = 1

; Instrument #1.
instr 1
  ; Generate a random number between 0 and 1.
  ; kalpha = 1

  il pcauchy 1

  print il
endin

; Instrument #2.
instr 2
  ; Generate a random number between 0 and 1.
  ; kalpha = 1

  seed 0

  il pcauchy 1

  print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  il = 0.012
```

## Voir Aussi

*seed, betarand, bexprnd, cauchy, exprand, gauss, linrand, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.

# pchbend

pchbend — Donne la valeur actuelle du pitch-bend pour ce canal.

## Description

Donne la valeur actuelle du pitch-bend pour ce canal.

## Syntaxe

```
ibend pchbend [imin] [, imax]
```

```
kbend pchbend [imin] [, imax]
```

## Initialisation

*imin*, *imax* (optionel) -- fixe les limites minimale et maximale pour les valeurs obtenues

## Exécution

Donne la valeur actuelle du pitch-bend pour ce canal. Noter que l'on a la valeur du pitch-bend qui est indépendant du pitch MIDI, ce qui permet d'utiliser cette valeur pour n'importe quel but.

## Exemples

Voici un exemple de l'opcode pchbend. Il utilise le fichier *pchbend.csd* [examples/pchbend.csd].

### Exemple 357. Exemple de l'opcode pchbend.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pchbend.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  il pchbend

  print il
endin
```



```
</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par Kevin Conder.

# pchmidi

pchmidi — Get the note number of the current MIDI event, expressed in pitch-class units.

## Description

Get the note number of the current MIDI event, expressed in pitch-class units.

## Syntax

ipch pchmidi

## Performance

Get the note number of the current MIDI event, expressed in pitch-class units for local processing.



### pchmidi vs. pchmidinn

The *pchmidi* opcode only produces meaningful results in a Midi-activated note (either real-time or from a Midi score with the -F flag). With *pchmidi*, the Midi note number value is taken from the Midi event that is internally associated with the instrument instance. On the other hand, the *pchmidinn* opcode may be used in any Csound instrument instance whether it is activated from a Midi event, score event, line event, or from another instrument. The input value for *pchmidinn* might for example come from a p-field in a textual score or it may have been retrieved from the real-time Midi event that activated the current note using the *notnum* opcode.

## Examples

Here is an example of the pchmidi opcode. It uses the file *pchmidi.csd* [examples/pchmidi.csd].

### Exemple 358. Example of the pchmidi opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac       -iadc      -d          -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pchmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
```

```
; This example expects MIDI note inputs on channel 1
il pchmidi

print i1
endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidib, veloc, cpsmidinn, octmidinn, pchmidinn*

## Credits

Author: Barry L. Vercoe - Mike Berry  
MIT - Mills  
May 1997

Example written by Kevin Conder.

# pchmidib

pchmidib — Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in pitch-class units.

## Description

Get the note number of the current MIDI event and modify it by the current pitch-bend value, express it in pitch-class units.

## Syntax

ipch **pchmidib** [irange]

kpch **pchmidib** [irange]

## Initialization

*irange* (optional) -- the pitch bend range in semitones

## Performance

Get the note number of the current MIDI event, modify it by the current pitch-bend value, and express the result in pitch-class units. Available as an i-time value or as a continuous k-rate value.

## Examples

Here is an example of the pchmidib pchmidib. It uses the file *pchmidib.csd* [examples/pchmidib.csd].

### Exemple 359. Example of the pchmidib pchmidib.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac        -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pchmidib.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; This example expects MIDI note inputs on channel 1
i1 pchmidib
```

```
    print i1
  endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, veloc*

## Credits

Author: Barry L. Vercoe - Mike Berry  
MIT - Mills  
May 1997

Example written by Kevin Conder.

# pchmidinn

pchmidinn — Convertit un numéro de note Midi en unités d'octave point classe de hauteur.

## Description

Convertit un numéro de note Midi en unités d'octave point classe de hauteur.

## Syntaxe

**pchmidinn** (MidiNoteNumber) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*pchmidinn* est une fonction qui prend une valeur de taux-i ou de taux-k représentant un numéro de note Midi et qui retourne la valeur de hauteur équivalente dans le format octave point classe de hauteur. Cette conversion suppose que le do médian (8.00 en *pch*) est la note Midi numéro 60. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.



### pchmidinn vs. pchmidi

L'opcode *pchmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un événement Midi, un événement de partition, un événement en ligne, ou depuis un autre instrument. La valeur d'entrée de *pchmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode *notnum* à partir de l'évènement Midi en temps-réel qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de taux-i ou de taux-k. D'un autre côté, l'opcode *pchmidi* ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option -F). Avec *pchmidi*, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

*pchmidinn* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 20. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *pchmidinn*. Il utilise le fichier *cpsmidinn.csd* [examples/cpsmidinn.csd].

### Exemple 360. Exemple de l'opcode *pchmidinn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
; i-time loop to print conversion table
imidiNN = 0
loop1:
  icps = cpsmidinn(imidiNN)
  ioct = octmidinn(imidiNN)
  ipch = pchmidinn(imidiNN)

  print imidiNN, icps, ioct, ipch

  imidiNN = imidiNN + 1
if (imidiNN < 128) igoto loop1
```

```
endin

instr 2
; test k-rate converters
kMiddleC = 60
kcps = cpsmidinn(kMiddleC)
koct = octmidinn(kMiddleC)
kpch = pchmidinn(kMiddleC)

printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*cpsmidinn, octmidinn, pchmidi, notnum, cpspch, cpsoct, octcps, octpch, pchoct*

## Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07



# pchoct

pchoct — Convertit une valeur octave-point-partie-décimale en classe de hauteur.

## Description

Convertit une valeur octave-point-partie-décimale en classe de hauteur.

## Syntaxe

**pchoct** (oct) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*pchoct* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 21. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *pchoct*. Il utilise le fichier *pchoct.csd* [examples/pchoct.csd].

### Exemple 361. Exemple de l'opcode *pchoct*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pchoct.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Convert an octave-point-decimal value into a
; pitch-class value.
ioct = 8.75
ipch = pchoct(ioct)

print ipch
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1: ipch = 8.090
```

## Voir Aussi

*cpsoct, cpspch, octcps, octpch, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Exemple écrit par Kevin Conder.

# pconvolve

convolve — Convolution basée sur un algorithme overlap-save à découpage uniforme.

## Description

Convolution basée sur un algorithme overlap-save à découpage uniforme. Comparé à l'opcode *convolve*, *pconvolve* a trois atouts :

- petit délai
- peut fonctionner en temps réel pour les fichiers de réponse impulsionnelle les plus courts
- pas de passe d'analyse avant le traitement
- restitution souvent plus rapide que *convolve*

## Syntaxe

```
ar1 [, ar2] [, ar3] [, ar4] pconvolve ain, ifilcod [, ipartitionsizesize, ichannel]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères définissant un fichier de réponse impulsionnelle. Les fichiers multi-canaux sont supportés. Le fichier doit avoir le même taux d'échantillonnage que l'orchestre. [Note : on ne peut pas utiliser les fichiers de *cvsanal* !] Il faut garder à l'esprit que les fichiers plus longs nécessitent plus de temps de calcul [et probablement une plus grande taille des fragments et plus de latence]. Avec les processeurs actuels, les fichiers dépassant quelques secondes pourront ne pas être restitués en temps réel.

*ipartitionsizesize* (facultatif, par défaut égal à la taille du tampon de sortie [-b]) -- la taille en échantillons de chaque morceau de la réponse impulsionnelle. C'est le paramètre qu'il faut ajuster pour avoir les meilleures performances en fonction de la taille du fichier de réponse impulsionnelle. En général, une petite taille signifie une latence moins importante mais plus de temps de calcul. Si l'on spécifie une valeur qui n'est pas une puissance de 2 l'opcode trouvera la plus petite puissance de 2 immédiatement supérieure et l'utilisera comme taille des fragments.

*ichannel* (facultatif) -- le canal de la réponse impulsionnelle à utiliser.

## Exécution

*ain* -- signal audio en entrée.

La latence totale de l'opcode peut être calculée comme ceci [*ipartitionsizesize* étant une puissance de 2]

```
ilatency = (ksmps < ipartitionsizesize ? ipartitionsizesize + ksmps : ipartitionsizesize)/sr
```

## Exemples

L'instrument 1 montre un exemple de convolution en temps réel.

L'instrument 2 montre comment faire une convolution basée sur un fichier avec une méthode de "prospection" pour supprimer tout délai.



## NOTE

Il faut télécharger les fichiers de réponse impulsionnelle depuis [noisevault.com](http://noisevault.com) ou remplacer les noms de fichier avec vos propres fichiers de réponse impulsionnelle.

```

sr = 44100
ksmps = 100
nchnls = 2

instr 1
kmix = .5 ; Wet/dry mix. Vary as desired.
kvol = .5*kmix ; Overall volume level of reverb. May need to adjust
; when wet/dry mix is changed, to avoid clipping.

; do some safety checking to make sure we the parameters a good
kmix = (kmix < 0 || kmix > 1 ? .5 : kmix)
kvol = (kvol < 0 ? 0 : .5*kvol*kmix)

; size of each convolution partion -- for best performance, this parameter needs to be tweaked
ipartitionsize = p4

; calculate latency of pconvolve opcode
idel = (ksmps < ipartitionsize ? ipartitionsize + ksmps : ipartitionsize)/sr
prints "Convoluting with a latency of %f seconds%n", idel

; actual processing
al, ar ins

awetl, awetr pconvolve kvol*(al+ar), "Mercedes-van.wav", ipartitionsize

; Delay dry signal, to align it with the convoled sig
adryl delay (1-kmix)*al, idel
adryr delay (1-kmix)*ar, idel

outs adryl+awetl, adryr+awetr
endin

instr 2
imix = 0.5 ; Wet/dry mix. Vary as desired.
ivol = .5*imix ; Overall volume level of reverb. May need to adjust
; when wet/dry mix is changed, to avoid clipping.

ipartitionsize = 32768 ; size of each convolution partion
idel = (ksmps < ipartitionsize ? ipartitionsize + ksmps : ipartitionsize)/sr ; latency of pconvolve

kcount init idel*kr

; since we are using a soundin [instead of ins] we can
; do a kind of "look ahead" by looping during one k-pass
; without output, creating zero-latency
loop:
al, ar soundin "John_Cage_1.aif", 0

awetl, awetr pconvolve ivol*(al+ar), "FactoryHall.aif", ipartitionsize

adryl delay (1-imix)*al, idel ; Delay dry signal, to align it with
adryr delay (1-imix)*ar, idel

kcount = kcount - 1
if kcount > 0 kgoto loop

outs awetl+adryl, awetr+adryr
endin

```

## Voir Aussi

*convolve, dconv.*

## Crédits

Auteur : Matt Ingalls  
2004

# pcount

pcount — Returns the number of pfields belonging to a note event.

## Description

*pcount* returns the number of pfields belonging to a note event.

## Syntax

icount **pcount**

## Initialization

*icount* - stores the number of pfields for the current note event.



### Note

Note that the reported number of pfields is not necessarily what's explicitly written in the score, but the pfields available to the instrument through mechanisms like *pfield carry*.

## Examples

Here is an example of the pcount opcode. It uses the file *pcount.csd* [examples/pcount.csd].

### Exemple 362. Example of the pcount opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      ; -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
;-o pcount.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Anthony Kozar Dec 2006
instr 1
  inum  pcount
  print inum
endin
</CsInstruments>
<CsScore>
i1  0 3 4 5      ; has 5 pfields
i1  1 3          ; has 5 due to carry
i1  2 3 4 5 6 7  ; has 7
e
</CsScore>
</CsoundSynthesizer>
```

The example will produce the following output:

```
SECTION 1:
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
instr 1:  inum = 5.000
B 0.000 .. 1.000 T 1.000 TT 1.000 M:      0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
instr 1:  inum = 5.000
B 1.000 .. 2.000 T 2.000 TT 2.000 M:      0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 7
instr 1:  inum = 7.000
```

The warnings occur because pfields are not used explicitly by the instrument.

## See Also

*pindex*

## Credits

Example by: Anthony Kozar

Dec. 2006



# pdclip

pdclip — Performs linear clipping on an audio signal or a phasor.

## Description

The *pdclip* opcode allows a percentage of the input range of a signal to be clipped to fullscale. It is similar to simply multiplying the signal and limiting the range of the result, but *pdclip* allows you to think about how much of the signal range is being distorted instead of the scalar factor and has a offset parameter for assymetric clipping of the signal range. *pdclip* is also useful for remapping phasors for phase distortion synthesis.

## Syntax

```
aout pdclip ain, kWidth, kCenter [, ibipolar [, ifullscale]]
```

## Initialization

*ibipolar* -- an optional parameter specifying either unipolar (0) or bipolar (1) mode. Defaults to unipolar mode.

*ifullscale* -- an optional parameter specifying the range of input and output values. The maximum will be *ifullscale*. The minimum depends on the mode of operation: zero for unipolar or *-ifullscale* for bipolar. Defaults to 1.0 -- you should set this parameter to the maximum expected input value.

## Performance

*ain* -- the input signal to be clipped.

*aout* -- the output signal.

*kWidth* -- the percentage of the signal range that is clipped (must be between 0 and 1).

*kCenter* -- an offset for shifting the unclipped window of the signal higher or lower in the range (essentially a DC offset). Values should be in the range [-1, 1] with a value of zero representing no shift (regardless of whether bipolar or unipolar mode is used).

The *pdclip* opcode performs linear clipping on the input signal *ain*. *kWidth* specifies the percentage of the signal range that is clipped. The rest of the input range is mapped linearly from zero to *ifullscale* in unipolar mode and from *-ifullscale* to *ifullscale* in bipolar mode. When *kCenter* is zero, equal amounts of the top and bottom of the signal range are clipped. A negative value shifts the unclipped range more towards the bottom of the input range and a positive value shifts it more towards the top. *ibipolar* should be 1 for bipolar operation and 0 for unipolar mode. The default is unipolar mode (*ibipolar* = 0). *ifullscale* sets the maximum amplitude of the input and output signals (defaults to 1.0).

This amounts to waveshaping the input with the following transfer function (normalized to *ifullscale*=1.0 in bipolar mode):

1| \_\_\_\_\_ x-axis is input range, y-axis is output  
 | /  
 | / width of clipped region is 2\*kWidth

```

-1    | 1    width of unclipped region is 2*(1 - kWidth)
-----|----- kCenter shifts the unclipped region
        | left or right (up to kWidth)
        |
        |
        |
-----|-1

```

Bipolar mode can be used for direct, linear distortion of an audio signal. Alternatively, unipolar mode is useful for modifying the output of a phasor before it is used to index a function table, effectively making this a phase distortion technique.

## See Also

*pdhalf, pdhalfy, limit, clip, distort1*

## Examples

Here is an example of the `pdclip` opcode. It uses the file *pdclip.csd* [examples/pdclip.csd].

### Exemple 363. Example of the `pdclip` opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; test instrument for pdclip opcode
instr 3

    idur      = p3
    iamp      = p4
    ifreq     = p5
    ifn       = p6

    kenv      linseg      0, .05, 1.0, idur - .1, 1.0, .05, 0
    aosc      oscil      1.0, ifreq, ifn

    kmod      expseg      0.00001, idur, 1.0
    aout      pdclip      aosc, kmod, 0.0, 1.0

                                out      kenv*aout*iamp

endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1
f2 0 16385 10 1 .5 .3333 .25 .5

; pdclipped sine wave
i3 0 3 15000 440 1
i3 + 3 15000 330 1
i3 + 3 15000 220 1
s

; pdclipped composite wave

```

```
i3 0 3 15000 440 2
i3 + 3 15000 330 2
i3 + 3 15000 220 2
e
</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Anthony Kozar  
January 2008

New in Csound version 5.08

# pdhalf

pdhalf — Distorts a phasor for reading the two halves of a table at different rates.

## Description

The *pdhalf* opcode is designed to emulate the "classic" phase distortion synthesis method of the Casio CZ-series of synthesizers from the mid-1980's. This technique reads the first and second halves of a function table at different rates in order to warp the waveform. For example, *pdhalf* can smoothly transform a sine wave into something approximating the shape of a saw wave.

## Syntax

```
aout pdhalf ain, kShapeAmount [, ibipolar [, ifullscale]]
```

## Initialization

*ibipolar* -- an optional parameter specifying either unipolar (0) or bipolar (1) mode. Defaults to unipolar mode.

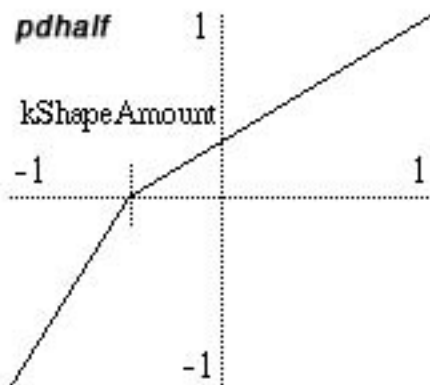
*ifullscale* -- an optional parameter specifying the range of input and output values. The maximum will be *ifullscale*. The minimum depends on the mode of operation: zero for unipolar or *-ifullscale* for bipolar. Defaults to 1.0 -- you should set this parameter to the maximum expected input value.

## Performance

*ain* -- the input signal to be distorted.

*aout* -- the output signal.

*kShapeAmount* -- the amount of distortion applied to the input. Must be between negative one and one (-1 to 1). An amount of zero means no distortion.



Transfer function created by *pdhalf* and a negative *kShapeAmount*.

The *pdhalf* opcode calculates a transfer function that is composed of two linear segments (see the graph). These segments meet at a "pivot point" which always lies on the same horizontal axis. (In unipolar mode, the axis is  $y = 0.5$ , and for bipolar mode it is the  $x$  axis). The *kShapeAmount* parameter speci-

fies where on the horizontal axis this point falls. When *kShapeAmount* is zero, the pivot point is in the middle of the input range, forming a straight line for the transfer function and thus causing no change in the input signal. As *kShapeAmount* changes from zero (0) to negative one (-1), the pivot point moves towards the left side of the graph, producing a phase distortion pattern like the Casio CZ's "sawtooth waveform". As it changes from zero (0) to positive one (1), the pivot point moves toward the right, producing an inverted pattern.

If the input to *pdhalf* is a phasor and the output is used to index a table, values for *kShapeAmount* that are less than zero will cause the first half of the table to be read more quickly than the second half. The reverse is true for values of *kShapeAmount* greater than zero. The rates at which the halves are read are calculated so that the frequency of the phasor is unchanged. Thus, this method of phase distortion can only produce higher partials in a harmonic series. It cannot produce inharmonic sidebands in the way that frequency modulation does.

*pdhalf* can work in either unipolar or bipolar modes. Unipolar mode is appropriate for signals like phasors that range between zero and some maximum value (selectable with *ifullscale*). Bipolar mode is appropriate for signals that range above and below zero by roughly equal amounts such as most audio signals. Applying *pdhalf* directly to an audio signal in this way results in a crude but adjustable sort of waveshaping/distortion.

A typical example of the use of *pdhalf* is

```

aphase    phasor    ifreq
apd       pdhalf    aphase, kamount
aout      tablei    apd, 1, 1

```

More information about Phase distortion synthesis can be found on Wikipedia at [http://en.wikipedia.org/wiki/Phase\\_distortion\\_synthesis](http://en.wikipedia.org/wiki/Phase_distortion_synthesis) [http://en.wikipedia.org/wiki/Phase\_distortion\_synthesis]

## See Also

*pdhalfy*, *pdclip*

## Examples

Here is an example of the *pdhalf* opcode. It uses the file *pdhalf.csd* [examples/pdhalf.csd].

### Exemple 364. Example of the *pdhalf* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; test instrument for pdhalf opcode
instr 4

    idur          = p3

```

```

iamp      = p4
ifreq     = p5
itable    = p6

aenv      linseg      0, .001, 1.0, idur - .051, 1.0, .05, 0
aosc      phasor      ifreq
kamount   linseg      0.0, 0.02, -0.99, 0.05, -0.9, idur-0.06, 0.0
apd       pdhalf      aosc, kamount
aout      tablei      apd, itable, 1

          out          aenv*aout*iamp

endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1
f2 0 16385 10 1 .5 .3333 .25 .5
f3 0 16385 9 1 1 270      ; inverted cosine

; descending "just blues" scale

; pdhalf with cosine table
; (imitates the CZ-101 "sawtooth waveform")
t 0 100
i4 0 .333 10000 512      3
i. + .      .      448
i. + .      .      384
i. + .      .      358.4
i. + .      .      341.33
i. + .      .      298.67
i. + 2      .      256
e
</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Anthony Kozar  
January 2008

New in Csound version 5.08

# pdhalfy

pdhalfy — Distorts a phasor for reading two unequal portions of a table in equal periods.

## Description

The *pdhalfy* opcode is a variation on the phase distortion synthesis method of the *pdhalf* opcode. It is useful for distorting a phasor in order to read two unequal portions of a table in the same number of samples.

## Syntax

```
aout pdhalfy ain, kShapeAmount [, ibipolar [, ifullscale]]
```

## Initialization

*ibipolar* -- an optional parameter specifying either unipolar (0) or bipolar (1) mode. Defaults to unipolar mode.

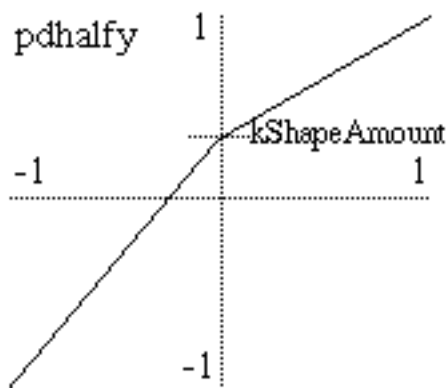
*ifullscale* -- an optional parameter specifying the range of input and output values. The maximum will be *ifullscale*. The minimum depends on the mode of operation: zero for unipolar or *-ifullscale* for bipolar. Defaults to 1.0 -- you should set this parameter to the maximum expected input value.

## Performance

*ain* -- the input signal to be distorted.

*aout* -- the output signal.

*kShapeAmount* -- the amount of distortion applied to the input. Must be between negative one and one (-1 to 1). An amount of zero means no distortion.



Transfer function created by *pdhalfy* and a negative *kShapeAmount*.

The *pdhalfy* opcode calculates a transfer function that is composed of two linear segments (see the graph). These segments meet at a "pivot point" which always lies on the same vertical axis. (In unipolar mode, the axis is  $x = 0.5$ , and for bipolar mode it is the  $y$  axis). So, *pdhalfy* is a variation of the *pdhalf*

opcode that places the pivot point of the phase distortion pattern on a vertical axis instead of a horizontal axis.

The *kShapeAmount* parameter specifies where on the vertical axis this point falls. When *kShapeAmount* is zero, the pivot point is in the middle of the output range, forming a straight line for the transfer function and thus causing no change in the input signal. As *kShapeAmount* changes from zero (0) to negative one (-1), the pivot point downward towards the bottom of the graph. As it changes from zero (0) to positive one (1), the pivot point moves upward, producing an inverted pattern.

If the input to *pdhalfy* is a phasor and the output is used to index a table, the use of *pdhalfy* will divide the table into two segments of different sizes with each segment being mapped to half of the oscillator period. Values for *kShapeAmount* that are less than zero will cause less than half of the table to be read in the first half of the period of oscillation. The rest of the table will be read in the second half of the period. The reverse is true for values of *kShapeAmount* greater than zero. Note that the frequency of the phasor is always unchanged. Thus, this method of phase distortion can only produce higher partials in a harmonic series. It cannot produce inharmonic sidebands in the way that frequency modulation does. *pdhalfy* tends to have a milder quality to its distortion than *pdhalf*.

*pdhalfy* can work in either unipolar or bipolar modes. Unipolar mode is appropriate for signals like phasors that range between zero and some maximum value (selectable with *ifullscale*). Bipolar mode is appropriate for signals that range above and below zero by roughly equal amounts such as most audio signals. Applying *pdhalfy* directly to an audio signal in this way results in a crude but adjustable sort of waveshaping/distortion.

A typical example of the use of *pdhalfy* is

```

aphase    phasor    ifreq
apd       pdhalfy   aphase, kamount
aout      tablei    apd, 1, 1

```

More information about Phase distortion synthesis can be found on Wikipedia at [http://en.wikipedia.org/wiki/Phase\\_distortion\\_synthesis](http://en.wikipedia.org/wiki/Phase_distortion_synthesis) [http://en.wikipedia.org/wiki/Phase\_distortion\_synthesis]

## See Also

*pdhalf*, *pdclip*

## Examples

Here is an example of the *pdhalfy* opcode. It uses the file *pdhalfy.csd* [examples/pdhalfy.csd].

### Exemple 365. Example of the *pdhalfy* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac        -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>

```



```

<CsInstruments>
; test instrument for pdhalfy opcode
instr 5

    idur      = p3
    iamp      = p4
    ifreq     = p5
    iamtinit  = p6      ; initial amount of phase distortion
    iatt      = p7      ; attack time
    isuslvl   = p8      ; sustain amplitude
    idistdec  = p9      ; time for distortion amount to reach zero
    itable    = p10

    idec      = idistdec - iatt
    irel      = .05
    isus      = idur - (idistdec + irel)

    aenv      linseg    0, iatt, 1.0, idec, isuslvl, isus, isuslvl, irel, 0, 0, 0
    kamount   linseg    -iamtinit, idistdec, 0.0, idur-idistdec, 0.0
    aosc      phasor    ifreq
    apd       pdhalfy   aosc, kamount
    aout      tablei    apd, itable, 1

                                out      aenv*aout*iamp

endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1                ; sine
f3 0 16385 9 1 1 270           ; inverted cosine

; descending "just blues" scale

; pdhalfy with cosine table
t 0 100
i5 0 .333 10000 512          1.0   .02  0.5  .12  3
i. + . . .                  448    <
i. + . . .                  384    <
i. + . . .                  358.4  <
i. + . . .                  341.33 <
i. + . . .                  298.67 <
i. + 2 . .                  256    0.5
s

; pdhalfy with sine table
t 0 100
i5 0 .333 10000 512          1.0   .001 0.1  .07  1
i. + . . .                  448    <
i. + . . .                  384    <
i. + . . .                  358.4  <
i. + . . .                  341.33 <
i. + . . .                  298.67 <
i. + 2 . .                  256    0.5
e
</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Anthony Kozar  
January 2008

New in Csound version 5.08

# peak

**peak** — Maintains the output equal to the highest absolute value received.

## Description

These opcodes maintain the output k-rate variable as the peak absolute level so far received.

## Syntax

```
kres peak asig
```

```
kres peak ksig
```

## Performance

*kres* -- Output equal to the highest absolute value received so far. This is effectively an input to the opcode as well, since it reads *kres* in order to decide whether to write something higher into it.

*ksig* -- k-rate input signal.

*asig* -- a-rate input signal.

## Examples

Here is an example of the peak opcode. It uses the file *peak.csd* [examples/peak.csd], and *beats.wav* [examples/beats.wav].

### Exemple 366. Example of the peak opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o peak.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
; Capture the highest amplitude in the "beats.wav" file.
asig soundin "beats.wav"
kp peak asig

; Print out the peak value once per second.
printk 1, kp
```

```
    out asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

Its output should include lines like this:

```
i 1 time 0.00002: 4835.00000
i 1 time 1.00002: 29312.00000
i 1 time 2.00002: 32767.00000
```

## Credits

Author: Robin Whittle  
Australia  
May 1997

Example written by Kevin Conder.

# peakk

peakk — Obsolète.

## Description

Obsolète depuis la version 3.63. Utiliser plutôt l'opcode *peak*.

# pgmassign

pgmassign — Affecte un numéro d'instrument à un numéro de programme MIDI spécifié.

## Description

Affecte un numéro d'instrument à un (ou à tous) le(s) programme(s) MIDI spécifié(s).

Par défaut, le numéro de l'instrument est le même que celui du programme. Si l'instrument choisi est inférieur ou égal à zéro, ou n'existe pas, le changement de programme est ignoré. Cet opcode est normalement utilisé dans l'en-tête de l'orchestre. Cependant, comme *massign*, il fonctionne aussi dans les instruments.

## Syntaxe

```
pgmassign ipgm, inst[, ichn]
```

```
pgmassign ipgm, "insname"[, ichn]
```

## Initialisation

*ipgm* -- numéro de programme MIDI (1 à 128). Une valeur de zéro sélectionne tous les programmes.

*inst* -- numéro d'instrument. S'il est inférieur ou égal à zéro, les changements de programme MIDI à *ipgm* sont ignorés. Actuellement, l'affectation à un instrument qui n'existe pas a le même effet. Ceci pourra changer dans une version future afin d'imprimer un message d'erreur.

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un nom d'instrument.

« *ichn* » (facultatif, par défaut zéro) -- numéro de canal. S'il vaut zéro, les changements de programme sont effectués sur tout les canaux.

Vous pouvez empêcher l'activation de n'importe quel instrument en utilisant l'en-tête ci-dessous :

```
massign 0, 0
pgmassign 0, 0
```

## Exemples

Voici un exemple de l'opcode pgmassign. Il utilise le fichier *pgmassign.csd* [examples/pgmassign.csd].

### Exemple 367. Exemple de l'opcode pgmassign.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc       -d          -M0   ;;RT audio I/O with MIDI in
```

```

; For Non-realtime ouput leave only the line below:
; -o pgmassign.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Program 55 (synth vox) uses Instrument #10.
pgmassign 55, 10

; Instrument #10.
instr 10
; Just an example, no working code in here!
endin

</CsInstruments>
<CsScore>

; Play Instrument #10 for one second.
i 10 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode `pgmassign` qui ignorera les évènements de changement de programme. Il utilise le fichier `pgmassign_ignore.csd` [examples/pgmassign\_ignore.csd].

### Exemple 368. Exemple de l'opcode `pgmassign` qui ignorera les évènements de changement de programme.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac        -iadc       -d           -M0 ;;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pgmassign_ignore.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Ignore all program change events.
pgmassign 0, -1

; Instrument #1.
instr 1
; Just an example, no working code in here!
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple avancé de l'opcode `pgmassign`. Il utilise le fichier `pgmassign_advanced.csd` [exemples/pgmassign\_advanced.csd].

Ne pas oublier qu'il faut inclure l'option `-F` lorsque l'on utilise un fichier MIDI externe comme « `pgmassign_advanced.mid` ».

### Exemple 369. Un exemple avancé de l'opcode `pgmassign`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d          -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pgmassign_advanced.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 1

    massign 1, 1 ; channels 1 to 4 use instr 1 by default
    massign 2, 1
    massign 3, 1
    massign 4, 1

; pgmassign.mid has 4 notes with these parameters:
;
;      Start time Channel Program
;
; note 1 0.5      1      10
; note 2 1.5      2      11
; note 3 2.5      3      12
; note 4 3.5      4      13

    pgmassign 0, 0      ; disable program changes
    pgmassign 11, 3      ; program 11 uses instr 3
    pgmassign 12, 2      ; program 12 uses instr 2

; waveforms for instruments
itmp ftgen 1, 0, 1024, 10, 1
itmp ftgen 2, 0, 1024, 10, 1, 0.5, 0.3333, 0.25, 0.2, 0.1667, 0.1429, 0.125
itmp ftgen 3, 0, 1024, 10, 1, 0, 0.3333, 0, 0.2, 0, 0.1429, 0, 0.10101

    instr 1      /* sine */

kcps cpsmidib 2 ; note frequency
asnd oscili 30000, kcps, 1
    out asnd

    endin

    instr 2      /* band-limited sawtooth */

kcps cpsmidib 2 ; note frequency
asnd oscili 30000, kcps, 2
    out asnd

    endin

    instr 3      /* band-limited square */

kcps cpsmidib 2 ; note frequency
asnd oscili 30000, kcps, 3
    out asnd

    endin

</CsInstruments>
<CsScore>
```

```
t 0 120
f 0 8.5 2 -2 0
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*midichn et massign*

## Crédits

Auteur : Istvan Varga  
Mai 2002

Nouveau dans la version 4.20



# phaser1

phaser1 — First-order allpass filters arranged in a series.

## Description

An implementation of *iord* number of first-order allpass filters in series.

## Syntax

```
ares phaser1 asig, kfreq, kord, kfeedback [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- used to control initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*kfreq* -- frequency (in Hz) of the filter(s). This is the frequency at which each filter in the series shifts its input by 90 degrees.

*kord* -- the number of allpass stages in series. These are first-order filters and can range from 1 to 4999.



### Note

Although *kord* is listed as k-rate, it is in fact accessed only at init-time. So if you are using a k-rate argument, it must be assigned with *init*.

*kfeedback* -- amount of the output which is fed back into the input of the allpass chain. With larger amounts of feedback, more prominent notches appear in the spectrum of the output. *kfeedback* must be between -1 and +1. for stability.

*phaser1* implements *iord* number of first-order allpass sections, serially connected, all sharing the same coefficient. Each allpass section can be represented by the following difference equation:

$$y(n) = C * x(n) + x(n-1) - C * y(n-1)$$

where  $x(n)$  is the input,  $x(n-1)$  is the previous input,  $y(n)$  is the output,  $y(n-1)$  is the previous output, and  $C$  is a coefficient which is calculated from the value of *kfreq*, using the bilinear z-transform.

By slowly varying *kfreq*, and mixing the output of the allpass chain with the input, the classic "phase shifter" effect is created, with notches moving up and down in frequency. This works best with *iord* between 4 and 16. When the input to the allpass chain is mixed with the output, 1 notch is generated for every 2 allpass stages, so that with *iord* = 6, there will be 3 notches in the output. With higher values for *iord*, modulating *kfreq* will result in a form of nonlinear pitch modulation.

## Examples

Here is an example of the phaser1 opcode. It uses the file *phaser1.csd* [examples/phaser1.csd].

### Exemple 370. Example of the phaser1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o phaser1.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; demonstration of phase shifting abilities of phaser1.
instr 1
; Input mixed with output of phaser1 to generate notches.
; Shows the effects of different iorder values on the sound
idur   = p3
iamp   = p4 * .05
iorder = p5          ; number of 1st-order stages in phaser1 network.
                        ; Divide iorder by 2 to get the number of notches.
ifreq  = p6          ; frequency of modulation of phaser1
ifeed  = p7          ; amount of feedback for phaser1

kamp   linseg 0, .2, iamp, idur - .2, iamp, .2, 0

iharms = (sr*.4) / 100

asig    gbuzz 1, 100, iharms, 1, .95, 2 ; "Sawtooth" waveform modulation oscillator for phaser1 ugen.
kfreq   oscili 5500, ifreq, 1
kmod    = kfreq + 5600

aphs    phaser1 asig, kmod, iorder, ifeed

out      (asig + apha) * iamp
endin

</CsInstruments>
<CsScore>

; inverted half-sine, used for modulating phaser1 frequency
f1 0 16384 9 .5 -1 0
; cosine wave for gbuzz
f2 0 8192 9 1 1 .25

; phaser1
i1 0 5 7000 4 .2 .9
i1 6 5 7000 6 .2 .9
i1 12 5 7000 8 .2 .9
i1 18 5 7000 16 .2 .9
i1 24 5 7000 32 .2 .9
i1 30 5 7000 64 .2 .9
e

</CsScore>
</CsoundSynthesizer>
```

## Technical History

A general description of the differences between flanging and phasing can be found in Hartmann [1]. An early implementation of first-order allpass filters connected in series can be found in Beigel [2], where the bilinear z-transform is used for determining the phase shift frequency of each stage. Cronin [3] presents a similar implementation for a four-stage phase shifting network. Chamberlin [4] and Smith [5] both discuss using second-order allpass sections for greater control over notch depth, width, and frequency.

## References

1. Hartmann, W.M. "Flanging and Phasers." Journal of the Audio Engineering Society, Vol. 26, No. 6, pp. 439-443, June 1978.
2. Beigel, Michael I. "A Digital 'Phase Shifter' for Musical Applications, Using the Bell Labs (Alles-Fischer) Digital Filter Module." Journal of the Audio Engineering Society, Vol. 27, No. 9, pp. 673-676, September 1979.
3. Cronin, Dennis. "Examining Audio DSP Algorithms." Dr. Dobb's Journal, July 1994, p. 78-83.
4. Chamberlin, Hal. Musical Applications of Microprocessors. Second edition. Indianapolis, Indiana: Hayden Books, 1985.
5. Smith, Julius O. "An Allpass Approach to Digital Phasing and Flanging." Proceedings of the 1984 ICMC, p. 103-108.

## See Also

*phaser2*

## Credits

Author: Sean Costello  
Seattle, Washington  
1999

November 2002. Added a note about the *kord* parameter, thanks to Rasmus Ekman.

New in Csound version 4.0

# phaser2

phaser2 — Second-order allpass filters arranged in a series.

## Description

An implementation of *iord* number of second-order allpass filters in series.

## Syntax

ares **phaser2** asig, kfreq, kq, kord, kmode, ksep, kfeedback

## Initialization

*iskip* (optional, default=0) -- used to control initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*kfreq* -- frequency (in Hz) of the filter(s). This is the center frequency of the notch of the first allpass filter in the series. This frequency is used as the base frequency from which the frequencies of the other notches are derived.

*kq* -- Q of each notch. Higher Q values result in narrow notches. A Q between 0.5 and 1 results in the strongest "phasing" effect, but higher Q values can be used for special effects.

*kord* -- the number of allpass stages in series. These are second-order filters, and *iord* can range from 1 to 2499. With higher orders, the computation time increases.

*kfeedback* -- amount of the output which is fed back into the input of the allpass chain. With larger amounts of feedback, more prominent notches appear in the spectrum of the output. *kfeedback* must be between -1 and +1. for stability.

*kmode* -- used in calculation of notch frequencies.



### Note

Although *kord* and *kmode* are listed as k-rate, they are in fact accessed only at init-time. So if you are using k-rate arguments, they must be assigned with *init*.

*ksep* -- scaling factor used, in conjunction with *imode*, to determine the frequencies of the additional notches in the output spectrum.

*phaser2* implements *iord* number of second-order allpass sections, connected in series. The use of second-order allpass sections allows for the precise placement of the frequency, width, and depth of notches in the frequency spectrum. *iord* is used to directly determine the number of notches in the spectrum; e.g. for *iord* = 6, there will be 6 notches in the output spectrum.

There are two possible modes for determining the notch frequencies. When *imode* = 1, the notch frequencies are determined the following function:

frequency of notch  $N = kbf + (ksep * kbf * N-1)$

For example, with  $imode = 1$  and  $ksep = 1$ , the notches will be in harmonic relationship with the notch frequency determined by  $kfreq$  (i.e. if there are 8 notches, with the first at 100 Hz, the next notches will be at 200, 300, 400, 500, 600, 700, and 800 Hz). This is useful for generating a "comb filtering" effect, with the number of notches determined by  $iord$ . Different values of  $ksep$  allow for inharmonic notch frequencies and other special effects.  $ksep$  can be swept to create an expansion or contraction of the notch frequencies. A useful visual analogy for the effect of sweeping  $ksep$  would be the bellows of an accordion as it is being played - the notches will be separated, then compressed together, as  $ksep$  changes.

When  $imode = 2$ , the subsequent notches are powers of the input parameter  $ksep$  times the initial notch frequency specified by  $kfreq$ . This can be used to set the notch frequencies to octaves and other musical intervals. For example, the following lines will generate 8 notches in the output spectrum, with the notches spaced at octaves of  $kfreq$ :

```
aphs phaser2 ain, kfreq, 0.5, 8, 2, 2, 0
aout =      ain + aphis
```

When  $imode = 2$ , the value of  $ksep$  must be greater than 0.  $ksep$  can be swept to create a compression and expansion of notch frequencies (with more dramatic effects than when  $imode = 1$ ).

## Examples

Here is an example of the phaser2 opcode. It uses the file *phaser2.csd* [examples/phaser2.csd].

### Exemple 371. Example of the phaser2 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o phaser2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 2
; demonstration of phase shifting abilities of phaser2.
; Input mixed with output of phaser2 to generate notches.
; Demonstrates the interaction of imode and ksep.
idur = p3
iamp = p4 * .04
iorder = p5 ; number of 2nd-order stages in phaser2 network
ifreq = p6 ; not used
ifeed = p7 ; amount of feedback for phaser2
imode = p8 ; mode for frequency scaling
isep = p9 ; used with imode to determine notch frequencies
kamp linseg 0, .2, iamp, idur - .2, iamp, .2, 0
iharms = (sr*.4) / 100
```

```

; "Sawtooth" waveform exponentially decaying function, to control notch frequencies
asig  gbuzz 1, 100, iharms, 1, .95, 2
kline  expseg 1, idur, .005
aphs  phaser2 asig, kline * 2000, .5, iorder, imode, isep, ifeed

out (asig + apha) * iamp
endin

</CsInstruments>
<CsScore>

; cosine wave for gbuzz
f2 0 8192 9 1 1 .25

; phaser2, imode=1
i2 00 10 7000 8 .2 .9 1 .33
i2 11 10 7000 8 .2 .9 1 2

; phaser2, imode=2
i2 22 10 7000 8 .2 .9 2 .33
i2 33 10 7000 8 .2 .9 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## Technical History

A general description of the differences between flanging and phasing can be found in Hartmann [1]. An early implementation of first-order allpass filters connected in series can be found in Beigel [2], where the bilinear z-transform is used for determining the phase shift frequency of each stage. Cronin [3] presents a similar implementation for a four-stage phase shifting network. Chamberlin [4] and Smith [5] both discuss using second-order allpass sections for greater control over notch depth, width, and frequency.

## References

1. Hartmann, W.M. "Flanging and Phasers." Journal of the Audio Engineering Society, Vol. 26, No. 6, pp. 439-443, June 1978.
2. Beigel, Michael I. "A Digital 'Phase Shifter' for Musical Applications, Using the Bell Labs (Alles-Fischer) Digital Filter Module." Journal of the Audio Engineering Society, Vol. 27, No. 9, pp. 673-676, September 1979.
3. Cronin, Dennis. "Examining Audio DSP Algorithms." Dr. Dobb's Journal, July 1994, p. 78-83.
4. Chamberlin, Hal. Musical Applications of Microprocessors. Second edition. Indianapolis, Indiana: Hayden Books, 1985.
5. Smith, Julius O. "An Allpass Approach to Digital Phasing and Flanging." Proceedings of the 1984 ICMC, p. 103-108.

## See Also

*phaser1*

## Credits

Author: Sean Costello  
Seattle, Washington  
1999

November 2002. Added a note about the *kord* and *kmode* parameters, thanks to Rasmus Ekman.

New in Csound version 4.0

# phasor

phasor — Produit une valeur de phase mobile normalisée.

## Description

Produit une valeur de phase mobile normalisée.

## Syntaxe

```
ares phasor xcps [ , iphs]
```

```
kres phasor kcps [ , iphs]
```

## Initialisation

*iphs* (facultatif) -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

## Exécution

Une phase interne est augmentée successivement selon la fréquence de *kcps* ou de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle  $0 \leq \text{phs} < 1$ .

Lorsqu'elle est utilisée comme indice dans une *table*, cette phase (multipliée par la longueur de la table de fonction) permettra de l'utiliser comme un oscillateur.

Noter que *phasor* est une sorte d'intégrateur, accumulant les incréments de phase qui représentent les réglages de fréquence.

## Exemples

Voici un exemple de l'opcode phasor. Il utilise le fichier *phasor.csd* [examples/phasor.csd].

### Exemple 372. Exemple de l'opcode phasor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o phasor.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```



```
; Instrument #1.
instr 1
; Create an index that repeats once per second.
kcps init 1
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kfreq table kndx, ifn, ixmode

; Generate a sine waveform, use our table values
; to vary its frequency.
a1 oscil 20000, kfreq, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a line from 200 to 2,000.
f 1 0 1025 -7 200 1024 2000
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

Les opcodes d'Accès aux Tables comme : *table*, *tablei*, *table3* et *tab*.

Aussi : *phasorbnk*.

## Crédits

Exemple écrit par Kevin Conder.

# phasorbnk

phasorbnk — Produit un nombre arbitraire de valeurs de phase mobiles normalisées.

## Description

Produit un nombre arbitraire de valeurs de phase mobiles normalisées, accessibles par un indice.

## Syntaxe

```
ares phasorbnk xcps, kndx, icnt [ , iphs]
```

```
kres phasorbnk kcps, kndx, icnt [ , iphs]
```

## Initialisation

*icnt* -- nombre maximum de phaseurs à utiliser.

*iphs* -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Si elle vaut -1, l'initialisation sera ignorée. Si *iphs* > 1 chaque phaseur sera initialisé avec une valeur aléatoire.

## Exécution

*kndx* -- valeur d'indice pour accéder aux phaseurs individuellement

Pour chaque phaseur indépendant, une phase interne est augmentée successivement selon la fréquence de *kcps* ou de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle  $0 \leq \text{phs} < 1$ . On accède à chaque phaseur individuel par l'indice *kndx*.

On peut utiliser cette banque de phaseurs dans une boucle de taux-k pour générer plusieurs voix indépendantes, ou en conjonction avec l'opcode *adsynt* pour changer les paramètres dans les tables utilisées par *adsynt*.

## Exemples

Voici un exemple de l'opcode phasorbnk. Il utilise le fichier *phasorbnk.csd* [examples/phasorbnk.csd].

### Exemple 373. Exemple de l'opcode phasorbnk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o phasorbnk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Generate a sinewave table.
giwave ftgen 1, 0, 1024, 10, 1

; Instrument #1
instr 1
; Generate 10 voices.
icnt = 10
; Empty the output buffer.
asum = 0
; Reset the loop index.
kindex = 0

; This loop is executed every k-cycle.
loop:
; Generate non-harmonic partials.
kcps = (kindex+1)*100+30
; Get the phase for each voice.
aphas phasorbnk kcps, kindex, icnt
; Read the wave from the table.
asig table aphas, giwave, 1
; Accumulate the audio output.
asum = asum + asig

; Increment the index.
kindex = kindex + 1

; Perform the loop until the index (kindex) reaches
; the counter value (icnt).
if (kindex < icnt) kgoto loop

out asum*3000
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Génère plusieurs voix avec des partiels indépendants. Cet exemple est meilleur avec *adsynt*. Voir aussi l'exemple de la notice *adsynt* pour une utilisation de *phasorbnk* au taux-k.

## Crédits

Auteur : Peter Neubäcker  
Munich, Allemagne  
Août 1999

Nouveau dans la version 3.58 de Csound

# pindex

pindex — Returns the value of a specified pfield.

## Description

*pindex* returns the value of a specified pfield.

## Syntax

```
ivalue pindex ipfieldIndex
```

## Initialization

*ipfieldIndex* - pfield number to query.

*ivalue* - value of the pfield.

## Examples

Here is an example of the pindex opcode. It uses the file *pindex.csd* [examples/pindex.csd].

### Exemple 374. Example of the pindex opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      ; -d          -M0    ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
;-o pindex.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Anthony Kozar Dec 2006

instr 1
  inum      pcount
  index     init 1
  loop1:
    ivalue  pindex index
    printf_i "p%d = %f\n", 1, index, ivalue
    index   = index + 1
    if (index <= inum) igoto loop1
  print inum
endin

</CsInstruments>
<CsScore>
i1  0 3 40 50          ; has 5 pfields
i1  1 2 80             ; has 5 due to carry
i1  2 1 40 50 60 70    ; has 7
e
</CsScore>
</CsoundSynthesizer>
```

The example will produce the following output:

```
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
p1 = 1.000000
p2 = 0.000000
p3 = 3.000000
p4 = 40.000000
p5 = 50.000000
instr 1: inum = 5.000
B 0.000 .. 1.000 T 1.000 TT 1.000 M: 0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
p1 = 1.000000
p2 = 1.000000
p3 = 2.000000
p4 = 80.000000
p5 = 50.000000
instr 1: inum = 5.000
B 1.000 .. 2.000 T 2.000 TT 2.000 M: 0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 7
p1 = 1.000000
p2 = 2.000000
p3 = 1.000000
p4 = 40.000000
p5 = 50.000000
p6 = 60.000000
p7 = 70.000000
instr 1: inum = 7.000
```

The warnings can be ignored, because the pfields are used indirectly through pindex instead of explicitly through p4, p5, etc.

## See Also

*pcount*

## Credits

Example by: Anthony Kozar

Dec. 2006

# pinkish

pinkish — Génère une approximation d'un bruit rose.

## Description

Génère une approximation d'un bruit rose (réponse à -3dB/oct) par une de ces deux méthodes :

- un générateur de bruit à taux multiples d'après Moore, codé par Martin Gardner
- un banc de filtres dessinés par Paul Kellet

## Syntaxe

```
ares pinkish xin [, imethod] [, inumbands] [, iseed] [, iskip]
```

## Initialisation

*imethod* (facultatif, par défaut=0) -- sélectionne la méthode de filtrage :

- 0 = méthode de Gardner (par défaut).
- 1 = banc de filtres de Kellet.
- 2 = Un banc de filtres quelque peu plus rapides par Kellet, avec une réponse moins précise.

*inumbands* (facultatif) -- ne fonctionne qu'avec la méthode de Gardner. Nombre de bandes de bruit à générer. Le maximum vaut 32 et le minimum vaut 4. Les valeurs plus élevées donnent un spectre plus lisse, mais au-delà de 20 bandes il y aura des fluctuations lentes presque comme une composante continue. La valeur par défaut est 20.

*iseed* (facultatif, par défaut=0) -- ne fonctionne qu'avec la méthode de Gardner. s'il est non nul, sert de graine au générateur de nombres aléatoires. S'il est nul, le générateur sera initialisé à partir de la valeur de l'horloge. Vaut 0 par défaut.

*iskip* (facultatif, par défaut=0) -- s'il est non nul, l'état interne n'est pas (ré)initialisé (utile pour les notes liées). Vaut 0 par défaut.

## Exécution

*xin* -- pour la méthode de Gardner : amplitude de taux-k ou -a. Pour les filtres de Kellet : normalement un bruit de taux-a de distribution uniforme obtenu à partir de *rand* (31-bit) ou de *unirand*, mais ça peut être n'importe quel signal de taux-a. La valeur de crête de la sortie varie largement ( $\pm 15\%$ ) même sur de longues périodes, et sera habituellement d'un niveau bien inférieur à celui de l'amplitude de l'entrée. Les valeurs de crête peuvent aussi dépasser occasionnellement l'amplitude de l'entrée ou celle du bruit.

*pinkish* tente de générer un bruit rose (c-à-d un bruit avec la même énergie dans chaque octave), par une des deux méthodes suivantes.

La première méthode, par Moore & Gardner, ajoute plusieurs signaux de bruit blanc (jusqu'à 32), géné-

rés à des taux en octave ( $sr$ ,  $sr/2$ ,  $sr/4$ , etc). Les valeurs pseudo aléatoires sont obtenues à partir d'un générateur interne sur 32 bit. Ce générateur est local à chaque instance de l'opcode et initialisable (comme pour *rand*).

La seconde méthode est un filtrage passe-bas avec une réponse d'environ -3dB/oct. Si l'entrée est un bruit blanc uniforme, la sortie sera un bruit rose. Avec cette méthode, on peut utiliser n'importe quel signal comme entrée. Le filtre de haute qualité est plus lent, mais il a moins d'ondulations et un intervalle de fréquences opératoires légèrement plus large que les versions moins gourmandes en calcul. Avec les filtres de Kellet, il n'y a pas de graine pour le générateur de nombres aléatoires.

La réponse en fréquence de la sortie dans la méthode de Gardner comporte quelques anomalies dans les intervalles basse-moyenne et moyenne-haute fréquence. On peut générer plus d'énergie en basse fréquence en augmentant le nombre de bandes. Cette méthode est aussi un peu plus rapide. Le filtre raffiné de Kellet a un spectre très lisse, mais un intervalle efficace plus limité. Le niveau augmente légèrement à l'extrémité haute du spectre.

## Exemples

Voici un exemple de l'opcode pinkish. Il utilise le fichier *pinkish.csd* [examples/pinkish.csd].

### Exemple 375. Exemple de l'opcode pinkish.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pinkish.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  awhite unirand 2.0

  ; Normalize to +/-1.0
  awhite = awhite - 1.0

  apink pinkish awhite, 1, 0, 0, 1

  out apink * 30000
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Bruit filtré (Kellet) pour une note liée (*iskip* est non nul).

## Crédits

Auteurs : Phil Burk et John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

Adapté pour Csound par Rasmus Ekman

La méthode par bandes de bruit est dûe à F. R. Moore (ou R. F. Voss), et fut présentée par Martin Gardner dans un article de *Scientific American* souvent cité. La présente version fut codée par Phil Burk après une discussion sur la liste de diffusion de music-dsp, avec des optimisations significatives suggérées par James McCartney.

Le banc de filtres a été dessiné par Paul Kellet, et posté sur la liste de diffusion de music-dsp.

La discussion complète sur le bruit rose a été rassemblée sur une page HTML par Robin Whittle, qui est actuellement consultable à <http://www.firstpr.com.au/dsp/pink-noise/>.

Notes ajoutées par Rasmus Ekman en Septembre 2002.



# pitch

pitch — Tracks the pitch of a signal.

## Description

Using the same techniques as *spectrum* and *specptrk*, pitch tracks the pitch of the signal in octave point decimal form, and amplitude in dB.

## Syntax

```
koct, kamp pitch asig, iupdte, ilo, ihi, idbthresh [, ifrqs] [, iconf] \  
[, istrtr] [, iocts] [, iq] [, inptls] [, irolloff] [, iskip]
```

## Initialization

*iupdte* -- length of period, in seconds, that outputs are updated

*ilo, ihi* -- range in which pitch is detected, expressed in octave point decimal

*idbthresh* -- amplitude, expressed in decibels, necessary for the pitch to be detected. Once started it continues until it is 6 dB down.

*ifrqs* (optional) -- number of divisions of an octave. Default is 12 and is limited to 120.

*iconf* (optional) -- the number of conformations needed for an octave jump. Default is 10.

*istrtr* (optional) -- starting pitch for tracker. Default value is  $(ilo + ihi)/2$ .

*iocts* (optional) -- number of octave decimations in spectrum. Default is 6.

*iq* (optional) -- Q of analysis filters. Default is 10.

*inptls* (optional) -- number of harmonics, used in matching. Computation time increases with the number of harmonics. Default is 4.

*irolloff* (optional) -- amplitude rolloff for the set of filters expressed as fraction per octave. Values must be positive. Default is 0.6.

*iskip* (optional) -- if non-zero, skips initialization. Default is 0.

## Performance

*koct* -- The pitch output, given in the octave point decimal format.

*kamp* -- The amplitude output.

*pitch* analyzes the input signal, *asig*, to give a pitch/amplitude pair of outputs, for the strongest frequency in the signal. The value is updated every *iupdte* seconds.

The number of partials and rolloff fraction can effect the pitch tracking, so some experimentation may be necessary. Suggested values are 4 or 5 harmonics, with rolloff 0.6, up to 10 or 12 harmonics with rolloff 0.75 for complex timbres, with a weak fundamental.

## Examples

Here is an example of the pitch opcode. It uses the file *pitch.csd* [examples/pitch.csd] and *mary.wav* [examples/mary.wav].

### Exemple 376. Example of the pitch opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pitch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - play an audio file without effects.
instr 1
  asig soundin "mary.wav"
  out asig
endin

; Instrument #2 - track the pitch of an audio file.
instr 2
  iupdt = 0.01
  ilo = 7
  ihi = 9
  idbthresh = 10
  ifrqs = 12
  iconf = 10
  istr = 8

  asig soundin "mary.wav"

  ; Follow the audio file, get its pitch and amplitude.
  koct, kamp pitch asig, iupdt, ilo, ihi, idbthresh, ifrqs, iconf, istr

  ; Re-synthesize the audio file with a different sounding waveform.
  kamp2 = kamp * 10
  kcps = cpsoct(koct)
  a1 oscil kamp2, kcps, 1

  out a1
endin

</CsInstruments>
<CsScore>

; Table #1: A different sounding waveform.
f 1 0 32768 11 7 3 .7

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
; Play Instrument #2, the "re-synthesized" waveform, for three seconds.
i 2 3 3
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: John ffitch  
University of Bath, Codemist Ltd.  
Bath, UK  
April 1999

Example written by Kevin Conder.

New in Csound version 3.54

# pitchamdf

pitchamdf — Follows the pitch of a signal based on the AMDF method.

## Description

Follows the pitch of a signal based on the AMDF method (Average Magnitude Difference Function). Outputs pitch and amplitude tracking signals. The method is quite fast and should run in realtime. This technique usually works best for monophonic signals.

## Syntax

```
kcps, krms pitchamdf asig, imincps, imaxcps [, icps] [, imedi] \  
[, idowns] [, iexcps] [, irmsmedi]
```

## Initialization

*imincps* -- estimated minimum frequency (expressed in Hz) present in the signal

*imaxcps* -- estimated maximum frequency present in the signal

*icps* (optional, default=0) -- estimated initial frequency of the signal. If 0,  $icps = (imincps + imaxcps) / 2$ . The default is 0.

*imedi* (optional, default=1) -- size of median filter applied to the output *kcps*. The size of the filter will be  $imedi * 2 + 1$ . If 0, no median filtering will be applied. The default is 1.

*idowns* (optional, default=1) -- downsampling factor for *asig*. Must be an integer. A factor of *idowns* > 1 results in faster performance, but may result in worse pitch detection. Useful range is 1 - 4. The default is 1.

*iexcps* (optional, default=0) -- how frequently pitch analysis is executed, expressed in Hz. If 0, *iexcps* is set to *imincps*. This is usually reasonable, but experimentation with other values may lead to better results. Default is 0.

*irmsmedi* (optional, default=0) -- size of median filter applied to the output *krms*. The size of the filter will be  $irmsmedi * 2 + 1$ . If 0, no median filtering will be applied. The default is 0.

## Performance

*kcps* -- pitch tracking output

*krms* -- amplitude tracking output

*pitchamdf* usually works best for monophonic signals, and is quite reliable if appropriate initial values are chosen. Setting *imincps* and *imaxcps* as narrow as possible to the range of the signal's pitch, results in better detection and performance.

Because this process can only detect pitch after an initial delay, setting *icps* close to the signal's real initial pitch prevents spurious data at the beginning.

The median filter prevents *kcps* from jumping. Experiment to determine the optimum value for *imedi* for a given signal.

Other initial values can usually be left at the default settings. Lowpass filtering of *asig* before passing it to *pitchamdf*, can improve performance, especially with complex waveforms.

## Examples

Here is an example of the *pitchamdf* opcode. It uses the file *pitchamdf.csd* [examples/pitchamdf.csd] and *mary.wav* [examples/mary.wav].

### Exemple 377. Example of the *pitchamdf* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d           ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pitchamdf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; synth waveform
giwave ftgen 2, 0, 1024, 10, 1, 1, 1, 1

; Instrument #1 - play an audio file with no effects.
instr 1
; get input signal with original freq.
asig soundin "mary.wav"

out asig
endin

; Instrument #2 - play the synth waveform using the
; same pitch and amplitude as the audio file.
instr 2
; get input signal with original freq.
asig soundin "mary.wav"

; lowpass-filter
asig tone asig, 1000
; extract pitch and envelope
kcps, krms pitchamdf asig, 150, 500, 200
; "re-synthesize" with the synth waveform, giwave.
asigl oscil krms, kcps, giwave

out asigl
endin

</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
; Play Instrument #2, the "re-synthesized" waveform, for three seconds.
i 2 3 3
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Peter Neubäcker  
Munich, Germany  
August 1999

New in Csound version 3.59

# planet

planet — Simulation d'une planète en orbite dans un système d'étoile binaire.

## Description

*planet* simule l'orbite d'une planète dans un système d'étoile binaire. Les sorties sont les coordonnées x, y et z de la planète en orbite. Il est possible que la planète atteigne sa vitesse de libération si elle croise une étoile de très près. Cela rend le système quelque peu instable.

## Syntaxe

```
ax, ay, az planet kmass1, kmass2, ksep, ix, iy, iz, ivx, ivy, ivz, idelta \
[, ifriction] [, iskip]
```

## Initialisation

*ix, iy, iz* -- les coordonnées initiales x, y et z de la planète

*ivx, ivy, ivz* -- les composantes initiales du vecteur vitesse de la planète.

*idelta* -- la taille du pas utilisé dans l'approximation de l'équation différentielle.

*ifriction* (facultatif, 0 par défaut) -- une valeur de frottement que l'on peut utiliser pour empêcher le système de diverger

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

## Exécution

*ax, ay, az* -- les coordonnées x, y et z de la planète en sortie

*kmass1* -- la masse de la première étoile

*kmass2* -- la masse de la seconde étoile

## Exemples

Voici un exemple de l'opcode *planet*. Il utilise le fichier *planet.csd* [examples/planet.csd].

### Exemple 378. Exemple de l'opcode planet.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
```

```

; -o planet.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1 - a planet orbiting in 3D space.
instr 1
; Create a basic tone.
kamp init 5000
kcps init 440
ifn = 1
asnd oscil kamp, kcps, ifn

; Figure out its X, Y, Z coordinates.
kml init 0.5
km2 init 0.35
ksep init 2.2
ix = 0
iy = 0.1
iz = 0
ivx = 0.5
ivy = 0
ivz = 0
ih = 0.0003
ifric = -0.1
ax1, ay1, az1 planet kml, km2, ksep, ix, iy, iz, \
                    ivx, ivy, ivz, ih, ifric

; Place the basic tone within 3D space.
kx downsamp ax1
ky downsamp ay1
kz downsamp az1
idist = 1
ift = 0
imode = 1
imdel = 1.018853416
iovr = 2
aw2, ax2, ay2, az2 spat3d asnd, kx, ky, kz, idist, \
                        ift, imode, imdel, iovr

; Convert the 3D sound to stereo.
aleft = aw2 + ay2
aright = aw2 - ay2

outs aleft, aright
endin

</CsInstruments>
<CsScore>

; Table #1 a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 10 seconds.
i 1 0 10
e

</CsScore>
</CsSoundSynthesizer>

```

## Crédits

Auteur : Hans Mikelson  
 Décembre 1998

Nouveau dans la version 3.50 de Csound



# pluck

pluck — Produit un son de corde pincée à décroissance naturelle ou un son de tambour.

## Description

La sortie audio est un son de corde pincée à décroissance naturelle ou un son de tambour basés sur l'algorithme de Karplus-Strong.

## Syntaxe

```
ares pluck kamp, kcps, icps, ifn, imeth [, iparm1] [, iparm2]
```

## Initialisation

*icps* -- valeur de hauteur attendue en Hz, utilisée pour fixer un tampon contenant une période d'échantillons audio qui sera lissée progressivement par une méthode de décroissance choisie. *icps* anticipe normalement la valeur de *kcps*, mais il peut recevoir artificiellement une grande ou une petite valeur pour influencer la taille du tampon d'échantillons.

*ifn* -- numéro de la table d'une fonction utilisée pour initialiser le tampon de décroissance cyclique. Si *ifn* = 0, une séquence aléatoire sera utilisée à la place.

*imeth* -- méthode de décroissance naturelle. Il y en a six, dont certaines utilisent les paramètres qui suivent.

1. moyenne simple. Un procédé de lissage simple, sans paramètres.
2. moyenne variable. Comme ci-dessus, avec une durée de lissage étirée d'un facteur de *iparm1* (=1 par défaut).
3. simple tambour. L'intervalle allant de la hauteur au bruit est contrôlé par un "facteur de rugosité" dans *iparm1* (0 à 1). Zéro donne l'effet de corde pincée, tandis que 1 inverse la polarité de chaque échantillon (baisse d'une octave, harmoniques impairs). La valeur 0.5 donne un son de caisse claire optimal.
4. tambour variable. Combine les facteurs de rugosité et d'étirement. *iparm1* est la rugosité (0 à 1), et *iparm2* est le facteur d'étirement (=1 par défaut).
5. moyenne pondérée. Comme la méthode 1, avec *iparm1* pondérant l'échantillon courant (le status quo) et *iparm2* pondérant l'échantillon précédant. *iparm1* + *iparm2* doit être <= 1.
6. filtre récursif du premier ordre, avec des coefficients de 0.5. N'est pas affecté par les paramètres.

*iparm1*, *iparm2* (facultatif) -- valeurs des paramètres à utiliser avec les algorithmes de lissage (ci-dessus). Les valeurs par défaut sont 0.

## Exécution

*kamp* -- l'amplitude de sortie.

*kcps* -- la fréquence de re-échantillonnage en Hz.

Un tampon audio interne, rempli lors de l'initialisation selon *ifn*, est continuellement re-échantillonné avec une fréquence de *kcps* et sa sortie est multipliée par *kamp*. Le re-échantillonnage du tampon est complété par un lissage pour simuler l'effet de décroissance naturelle du son.

Les cordes pincées (1, 2, 5, 6) sont plus réalistes si l'on commence avec une source de bruit, qui est riche en harmoniques initiaux. Les sons de tambour (méthodes 3 et 4) fonctionnent mieux avec une source plate (impulsion large), qui produit une attaque très bruiteuse et une extinction rapide.

L'algorithme original de Karplus-Strong utilisait un nombre fixe d'échantillons par cycle, ce qui provoquait une sérieuse quantification des hauteurs disponibles et de leur intonation. Cette implémentation re-échantillonne un tampon à la hauteur exacte donnée par *kcps*, qui peut être variée pour des effets de vibrato ou de glissando. Avec de faibles valeurs du taux d'échantillonnage de l'orchestre (par exemple *sr* = 10000), les fréquences élevées ne stockeront que très peu d'échantillons (*sr* / *icps*). Comme ceci peut causer un bruit notable lors du re-échantillonnage, le tampon interne a une taille minimale de 64 échantillons. Celui-ci peut être agrandi en fixant *icps* à une hauteur artificiellement basse.

## Exemples

Voici un exemple de l'opcode *pluck*. Il utilise le fichier *pluck.csd* [examples/pluck.csd].

### Exemple 379. Exemple de l'opcode *pluck*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pluck.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 20000
  kcps = 440
  icps = 440
  ifn = 0
  imeth = 1

  a1 pluck kamp, kcps, icps, ifn, imeth
  out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Exemple écrit par Kevin Conder.

# poisson

poisson — Générateur de nombres aléatoires de distribution de Poisson (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution de Poisson (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares poisson klambda
```

```
ires poisson klambda
```

```
kres poisson klambda
```

## Exécution

*ares*, *kres*, *ires* - nombre d'évènements se produisant (toujours un entier).

*klambda* - le nombre attendu d'évènements par intervalle d'échantillonnage.

## Adapté de Wikipédia :

En théorie des probabilités et en statistiques, la distribution de Poisson est une distribution de probabilité discrète. Elle exprime la probabilité d'apparition d'un certain nombre d'évènements pendant une période de temps fixée si ces évènements se produisent avec un taux moyen connu et indépendamment du temps écoulé depuis le dernier évènement.

La distribution de Poisson décrivant la probabilité qu'il y ait exactement  $k$  évènements ( $k$  étant un nombre non négatif,  $k = 0, 1, 2, \dots$ ) est :

$$f(k; \lambda) = \frac{e^{-\lambda} \lambda^k}{k!},$$

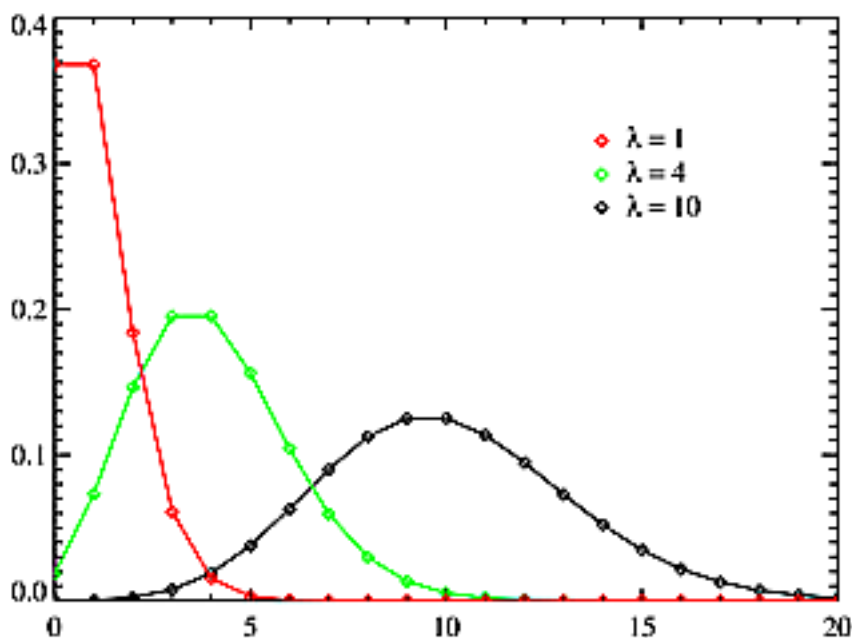
où :

- $\lambda$  est un nombre réel positif, égal au nombre attendu d'évènements se produisant durant l'intervalle donné. Par exemple, si les évènements se produisent en moyenne toutes les 4 minutes, et que l'on est intéressé par le nombre d'évènements se produisant dans un intervalle de 10 minutes, on utilisera comme modèle une distribution de Poisson avec  $\lambda = 10/4 = 2,5$ . Ce paramètre se nomme *klambda* dans les opcodes *poisson*.
- $k$  fait référence au nombre de i-, k- ou a- périodes écoulées.

La distribution de Poisson apparaît aussi avec les processus de Poisson. Elle s'applique à différents phénomènes de nature discrète (c-à-d, ceux qui peuvent se produire 0, 1, 2, 3, ... fois durant une période de temps donnée ou dans un espace donné) chaque fois que la probabilité du phénomène se produisant est constante dans le temps ou dans l'espace. Parmi les exemples qui peuvent être modélisés par une distri-

bution de Poisson, on trouve :

- Le nombre d'automobiles passant devant un repère sur une route (suffisamment éloigné des feux de circulation) pendant un intervalle de temps donné.
- Le nombre de fautes de frappe que l'on fait lorsque l'on tape une page.
- Le nombre d'appels par minute dans un centre d'appel téléphonique.
- Le nombre d'accès par minute à un serveur web.
- Le nombre d'animaux écrasés par unité de longueur sur une route.
- Le nombre de mutations dans un brin d'ADN après une certaine quantité de radiations.
- Le nombre de noyaux instables qui a diminué pendant une période de temps donnée dans un morceau de substance radioactive. Comme la radioactivité de la substance diminue avec le temps, l'intervalle de temps total utilisé dans le modèle doit être significativement inférieur à la durée de vie moyenne de la substance.
- Le nombre de pins par unité de surface dans une forêt hétérogène.
- Le nombre d'étoiles dans une région donnée de l'espace.
- La distribution des cellules réceptrices de la vision dans la rétine de l'oeil humain.
- Le nombre de virus qui peuvent infecter une cellule dans une culture de cellules.



Un diagramme montrant la distribution de Poisson.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286

2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode poisson. Il utilise le fichier *poisson.csd* [exemples/poisson.csd]. Il est écrit pour des systèmes \*NIX et génèrera des erreurs sur Windows.

### Exemple 380. Exemple de l'opcode poisson.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o poisson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 441 ;ksmps set deliberately high to have few k-periods per second
nchnls = 1

; Instrument #1.
instr 1
; Generates a random number in a poisson distribution.
; klambda = 1

i1 poisson 1

print i1
endin

instr 2

kres poisson p4
printk (ksmps/sr),kres ;prints every k-period
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
i 2 1 0.2 0.5
i 2 2 0.2 4 ;average 4 events per k-period
i 2 3 0.2 20 ;average 20 events per k-period
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*seed, betarand, bexpnrnd, cauchy, exprand, gauss, linrand, pcauchy, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder et Andrés Cabrera

# polyaft

polyaft — Retourne la pression d'after-touch polyphonique du numéro de note sélectionné.

## Description

*polyaft* retourne la pression polyphonique du numéro de note choisi, optionnellement mappé dans un intervalle défini par l'utilisateur.

## Syntaxe

```
ires polyaft inote [, ilow] [, ihigh]
```

```
kres polyaft inote [, ilow] [, ihigh]
```

## Initialisation

*inote* -- numéro de note. Normalement ajusté à la valeur retournée par *notnum*

*ilow* (facultatif, par défaut : 0) -- la valeur de sortie la plus basse

*ihigh* (facultatif, par défaut : 127) -- la valeur de sortie la plus haute

## Exécution

*kres* -- Pression polyphonique (aftertouch).

## Exemples

Voici un exemple de l'opcode *polyaft*. Il utilise le fichier *polyaft.csd* [examples/polyaft.csd].

Ne pas oublier d'inclure l'option *-F* lorsque l'on utilise un fichier MIDI externe comme « polyaft.mid ».

### Exemple 381. Exemple de l'opcode *polyaft*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac      -iadc      -d      -M0      ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o polyaft.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 1

    massign 1, 1
```



```

itmp ftgen 1, 0, 1024, 10, 1           ; sine wave

    instr 1

kcps cpsmidib 2           ; note frequency
inote notnum           ; note number
kaft polyaft inote, 0, 127 ; aftertouch
    ; interpolate aftertouch to eliminate clicks
ktmp phasor 40
ktmp trigger 1 - ktmp, 0.5, 0
kaft tlineto kaft, 0.025, ktmp
    ; map to sine curve for crossfade
kaft = sin(kaft * 3.14159 / 254) * 22000

asnd oscili kaft, kcps, 1

    out asnd

    endin

</CsInstruments>
<CsScore>

t 0 120
f 0 9 2 -2 0
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ajouté grâce à un courriel de Istvan Varga

Nouveau dans la version 4.12

# polynomial

polynomial — Efficiently evaluates a polynomial of arbitrary order.

## Description

The *polynomial* opcode calculates a polynomial with a single a-rate input variable. The polynomial is a sum of any number of terms in the form  $kn \cdot x^n$  where  $kn$  is the  $n$ th coefficient of the expression. These coefficients are k-rate values.

## Syntax

```
aout polynomial ain, k0 [, k1 [, k2 [...]]]
```

## Performance

*ain* -- the input signal used as the independent variable of the polynomial ("x").

*aout* -- the output signal ("y").

*k0*, *k1*, *k2*, ... -- the coefficients for each term of the polynomial.

If we consider the input parameter *ain* to be "x" and the output *aout* to be "y", then the *polynomial* opcode calculates the following equation:

$$y = k0 + k1 \cdot x + k2 \cdot x^2 + k3 \cdot x^3 + \dots$$

## See Also

*chebyshevpoly*, *mac maca*

## Examples

Here is an example of the polynomial opcode. It uses the file *polynomial.csd* [examples/polynomial.csd].

### Exemple 382. Example of the polynomial opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; no sound output
-n
</CsOptions>
<CsInstruments>

sr = 44100 ; audio rate is not important
```

```

kr = 10 ; execute the statements in instr 1 ten times per second

instr 1
    ; ax will vary from 1 to 10
    ax      init      1

    ; ay = ax^3 + 2ax^2 + 3ax + 4
    ay      polynomial ax, 4, 3, 2, 1

    ; convert our a-rate signals to k-rate values so that we can print
    ky      downsamp  ay
    kx      downsamp  ax
    printf  "%d:  %d\n", kx, kx, ky

    ax      =          ax + 1
endin

</CsInstruments>
<CsScore>

i1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Anthony Kozar  
January 2008

New in Csound version 5.08

# pop

push — Pops values from the global stack.

## Description

Pops values from the global stack.

## Syntax

```
xval1, [xval2, ... , xval31] pop
```

```
ival1, [ival2, ... , ival31] pop
```

## Initialization

*ival1 ... ival31* - values to be popped from the stack.

## Performance

*xval1 ... xval31* - values to be popped from the stack.

The given values are popped from the stack. The global stack works in LIFO order: after multiple *push* calls, *pop* should be used in reverse order.

Each *push* or *pop* operation can work on a "bundle" of multiple variables. When using *pop*, the number, type, and order of items must match those used by the corresponding *push*. That is, after a 'push Sfoo, ibar', you must call something like 'pop Sbar, ifoo', and not e.g. two separate 'pop' statements.

*push* and *pop* opcodes can take variables of any type (i-, k-, a- and strings). Use of any combination of i, k, a, and S types is allowed. Variables of type 'a' and 'k' are passed at performance time only, while 'i' and 'S' are passed at init time only.

push/pop for a, k, i, and S types copy data by value. By contrast, *push\_f* only pushes a "reference" to the f-signal, and then the corresponding *pop\_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push\_f* before *pop\_f* is called is not recommended, and if the instrument instance owning the variable that was passed by *push\_f* is deactivated before *pop\_f* is called, undefined behavior may occur.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

## See also

*stack*, *push*, *pop\_f* and *push\_f*.

## Credits

By: Istvan Varga.

2006

# pop\_f

pop\_f — Pops an f-sig frame from the global stack.

## Description

Pops an f-sig frame from the global stack.

## Syntax

`f sig pop_f`

## Performance

*f*sig - f-signal to be popped from the stack.

The values are popped the stack. The global stack must be initialized before used, and its size must be set. The global stack works in LIFO order: after multiple *push\_f* calls, *pop\_f* should be used in reverse order.

push/pop for a, k, i, and S types copy data by value. By contrast, *push\_f* only pushes a "reference" to the f-signal, and then the corresponding *pop\_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push\_f* before *pop\_f* is called is not recommended, and if the instrument instance owning the variable that was passed by *push\_f* is deactivated before *pop\_f* is called, undefined behavior may occur.

*push\_f* and *pop\_f* can only take a single argument, and the data is passed both at init and performance time.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

## See also

*stack*, *push*, *pop* and *push\_f*.

## Credits

By: Istvan Varga.

2006

# port

port — Applique un portamento à un signal de contrôle en escalier.

## Description

Applique un portamento à un signal de contrôle en escalier.

## Syntaxe

```
kres port ksig, ihtim [, isig]
```

## Initialisation

*ihtim* -- durée à mi-parcours de la fonction, en secondes.

*isig* (facultatif, par défaut 0) -- valeur initiale (c-à-d. précédente) pour la rétroaction interne. La valeur par défaut est 0. Avec une valeur négative l'initialisation sera ignorée et la dernière valeur de l'instance précédente sera la valeur initiale de la note.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*port* applique un portamento à un signal de contrôle en escalier. A chaque nouveau palier, *ksig* est filtré par un filtre passe-bas pour que la transition vers cette valeur se fasse au taux déterminé par *ihtim*. *ihtim* est la durée à « mi-parcours » de la fonction (en secondes), au cours de laquelle la courbe parcourera la moitié de la distance la séparant de la nouvelle valeur, puis la moitié de la moitié, etc., n'atteignant théoriquement jamais son asymptote. Avec *portk*, la durée à mi-parcours peut être variée au taux de contrôle.

## Voir Aussi

*areson*, *aresonk*, *atone*, *atonek*, *portk*, *reson*, *resonk*, *tone*, *tonek*

# portk

portk — Applique un portamento à un signal de contrôle en escalier.

## Description

Applique un portamento à un signal de contrôle en escalier.

## Syntaxe

```
kres portk ksig, khtim [, isig]
```

## Initialisation

*isig* (facultatif, par défaut 0) -- valeur initiale (c-à-d. précédente) pour la rétroaction interne. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*khtim* -- durée à mi-parcours de la fonction, en secondes.

*portk* est semblable à *port* à part le fait que la durée à mi-parcours peut-être variée au taux de contrôle.

## Exemples

Voici un exemple de l'opcode portk. Il utilise le fichier *portk.csd* [examples/portk.csd].

### Exemple 383. Exemple de l'opcode portk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            ; -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o portk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1

;Example by Andres Cabrera 2007

FLpanel "Slider", 650, 140, 50, 50
gkval1, gislider1 FLslider "Watch me", 0, 127, 0, 5, -1, 580, 30, 25, 20
gkval2, gislider2 FLslider "Move me", 0, 127, 0, 5, -1, 580, 30, 25, 80
```

```
gkhtim, gislider3 FLslider "khtim", 0.1, 1, 0, 6, -1, 30, 100, 610, 10
FLpanelEnd
FLrun

FLsetVal_i 0.1, gislider3 ;set initial time to 0.1

instr 1
kval portk gkval2, gkhtim ; take the value of slider 2 and apply portamento
FLsetVal 1, kval, gislider1 ;set the value of slider 1 to kval
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute.
i 1 0 60
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*areson, aresonk, atone, atonek, port, reson, resonk, tone, tonek*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997



# poscil

poscil — Oscillateur haute précision.

## Description

Oscillateur haute précision.

## Syntaxe

```
ares poscil aamp, acps, ifn [, iphs]
```

```
ares poscil aamp, kcps, ifn [, iphs]
```

```
ares poscil kamp, acps, ifn [, iphs]
```

```
ares poscil kamp, kcps, ifn [, iphs]
```

```
ires poscil kamp, kcps, ifn [, iphs]
```

```
kres poscil kamp, kcps, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction

*iphs* (facultatif, par défaut 0) -- phase initiale (table normalisée, index 0-1)

## Exécution

*ares* -- signal de sortie

*kamp*, *aamp* -- l'amplitude du signal de sortie.

*kcps*, *acps* -- la fréquence du signal de sortie en cycles par seconde.

*poscil* (oscillateur de précision) est identique à *oscili*, mais il permet un contrôle de la fréquence plus précis, en particulier lorsque l'on utilise de grandes tables avec de faibles valeurs de fréquence. Il utilise une indexation de la table en virgule flottante, au lieu de l'arithmétique entière utilisée par *oscil* et *oscili*. Il est à peine plus lent que *oscili*.

Depuis Csound 4.22, *poscil* accepte aussi des valeurs de fréquence négatives et il peut utiliser des valeurs de taux-a aussi bien pour l'amplitude que pour la fréquence. Ainsi, cet opcode permet la modulation d'amplitude (MA) et la modulation de fréquence (MF).

L'opcode *poscil3* est le même que *poscil*, mais il utilise une interpolation cubique.

Noter que *poscil* peut utiliser des tables de longueur différée (non puissance de deux).

## Exemples

Voici un exemple de l'opcode `poscil`. Il utilise le fichier `poscil.csd` [examples/poscil.csd].

### Exemple 384. Exemple de l'opcode `poscil`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o poscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  al poscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*poscil3*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1998

Exemple écrit par Kevin Conder.

Novembre 2002. Ajout d'une note sur les changements dans la version 4.22 de Csound, merci à Rasmus Ekman.

Nouveau dans la version 3.52 de Csound

# poscil3

poscil3 — Oscillateur haute précision avec interpolation cubique.

## Description

Oscillateur haute précision avec interpolation cubique.

## Syntax

```
ares poscil3 kamp, kcps, ifn [, iphs]
```

```
kres poscil3 kamp, kcps, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction

*iphs* (facultatif, par défaut 0) -- phase initiale (table normalisée, index 0-1)

## Exécution

*ares* -- signal de sortie

*kamp* -- amplitude du signal de sortie.

*kcps* -- fréquence du signal de sortie en cycles par seconde.

*poscil3* fonctionne comme *poscil*, mais il utilise l'interpolation cubique.

Noter que *poscil3* peut utiliser des tables de longueur différée (non puissance de deux).

## Exemples

Voici un exemple de l'opcode *poscil3*. Il utilise le fichier *poscil3.csd* [exemples/poscil3.csd].

### Exemple 385. Exemple de l'opcode *poscil3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o poscil3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```

kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  al poscil3 kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode poscil3, qui utilise une table remplie à partir d'un fichier son. Il utilise le fichier *poscil3-file.csd* [examples/poscil3-file.csd].

### Exemple 386. Un autre exemple de l'opcode poscil3.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o poscil3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  al poscil3 kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*poscil*

## Crédits

Auteurs : John ffitich, Gabriel Maldonado  
Italie

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.52 de Csound

# pow

pow — Calcule l'élévation à la puissance d'un argument par l'autre argument.

## Description

Calcule *xarg* élevé à la puissance *kpow* (ou *ipow*) et pondère le résultat par *inorm*.

## Syntaxe

```
ares pow aarg, kpow [, inorm]
```

```
ires pow iarg, ipow [, inorm]
```

```
kres pow karg, kpow [, inorm]
```

## Initialisation

*inorm* (facultatif, par défaut=1) -- Le nombre qui divisera le résultat (1 par défaut). Particulièrement utile si l'on calcule des puissances de signaux de taux -a ou de taux -k, ce qui produit très souvent des échantillons hors intervalle.

## Exécution

*aarg*, *iarg*, *karg* -- la base.

*ipow*, *kpow* -- l'exposant.



### Note

Utiliser ^ avec précaution dans les instructions arithmétiques, car les règles de précedence peuvent ne pas être correctes. Nouveau dans la version 3.493 de Csound.

## Exemples

Voici un exemple de l'opcode pow. Il utilise le fichier *pow.csd* [examples/pow.csd].

### Exemple 387. Exemple de l'opcode pow.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pow.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; This could also be expressed as: i1 = 2 ^ 12
i1 pow 2, 12

print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1: i1 = 4096.000
```

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.



# powershape

powershape — Waveshapes a signal by raising it to a variable exponent.

## Description

The *powershape* opcode raises an input signal to a power with pre- and post-scaling of the signal so that the output will be in a predictable range. It also processes negative inputs in a symmetrical way to positive inputs, calculating a dynamic transfer function that is useful for waveshaping.

## Syntax

```
aout powershape ain, kShapeAmount [, ifullscale]
```

## Initialization

*ifullscale* -- optional parameter specifying the range of input values from *-ifullscale* to *ifullscale*. Defaults to 1.0 -- you should set this parameter to the maximum expected input value.

## Performance

*ain* -- the input signal to be shaped.

*aout* -- the output signal.

*kShapeAmount* -- the amount of the shaping effect applied to the input; equal to the power that the input signal is raised.

The *powershape* opcode is very similar to the *pow* unit generators for calculating the mathematical "power of" operation. However, it introduces a couple of twists that can make it much more useful for waveshaping audio-rate signals. The *kShapeAmount* parameter is the exponent to which the input signal is raised.

To avoid discontinuities, the *powershape* opcode treats all input values as positive (by taking their absolute value) but preserves their original sign in the output signal. This allows for smooth shaping of any input signal while varying the exponent over any range. (*powershape* also (hopefully) deals intelligently with discontinuities that could arise when the exponent and input are both zero. Note though that negative exponents will usually cause the signal to exceed the maximum amplitude specified by the *ifullscale* parameter and should normally be avoided).

The other adaptation involves the *ifullscale* parameter. The input signal is divided by *ifullscale* before being raised to *kShapeAmount* and then multiplied by *ifullscale* before being output. This normalizes the input signal to the interval [-1,1], guaranteeing that the output (before final scaling) will also be within this range for positive shaping amounts and providing a smoothly evolving transfer function while varying the amount of shaping. Values of *kShapeAmount* between (0,1) will make the signal more "convex" while values greater than 1 will make it more "concave". A value of exactly 1.0 will produce no change in the input signal.

## See Also

*pow*, *powoftwo*

## Examples

Here is an example of the powershape opcode. It uses the file *powershape.csd* [examples/power-shape.csd].

### Exemple 388. Example of the powershape opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
instr 1
    imaxamp      =      10000
    kshapeamt    line    p5, p3, p6
    aosc         oscili   1.0, cpspch(p4), 1
    aout         powershape aosc, kshapeamt
    adeclick     linseg   0.0, 0.01, 1.0, p3 - 0.06, 1.0, 0.05, 0.0

                                out      aout * adeclick * imaxamp
endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1

i1 0 1      7.00  0.000001 0.8
i1 + 0.5    7.02  0.01    1.0
i1 + .      7.05  0.5      1.0
i1 + .      7.07  4.0      1.0
i1 + .      7.09  1.0     10.0
i1 + 2      7.06  1.0     25.0

e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Anthony Kozar

January 2008

New in Csound version 5.08

# powoftwo

powoftwo — Calcule une puissance de deux.

## Description

Calcule une puissance de deux.

## Syntaxe

`powoftwo(x)` (argument au taux d'initialisation ou de contrôle seulement)

## Exécution

La fonction *powoftwo()* retourne  $2^x$  et accepte comme argument des nombres positifs et négatifs. L'intervalle des valeurs autorisées dans *powoftwo()* va de -5 à +5 permettant une précision plus fine qu'un cent dans un intervalle de dix octaves. Pour un intervalle de valeurs plus grand, utiliser l'opcode plus lent *pow*.

Ces fonctions sont rapides, car elles lisent des valeurs stockées dans des tables. Elles sont très utiles lorsque l'on travaille avec des rapports de hauteurs. Elles travaillent au taux-i et au taux-k.

## Exemples

Voici un exemple de l'opcode powoftwo. Il utilise le fichier *powoftwo.csd* [exemples/powoftwo.csd].

### Exemple 389. Exemple de l'opcode powoftwo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o powoftwo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = powoftwo(12)
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
```

```
i 1 0 1  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 4096.000
```

## Voir Aussi

*logbtwo, pow*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Juin 1998

Auteur : John ffitch  
Université de Bath, Codemist, Ltd.  
Bath, UK  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# prealloc

prealloc — Crée de l'espace pour des instruments mais ne les exécute pas.

## Description

Crée de l'espace pour des instruments mais ne les exécute pas.

## Syntaxe

```
prealloc insnum, icount
```

```
prealloc "insname", icount
```

## Initialisation

*insnum* -- numéro de l'instrument

*icount* -- nombre d'allocations de l'instrument

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un instrument nommé.

## Exécution

Toutes les instances de *prealloc* doivent être définies dans la section d'en-tête, pas dans le corps de l'instrument.

## Exemples

Voici un exemple de l'opcode *prealloc*. Il utilise le fichier *prealloc.csd* [examples/prealloc.csd].

### Exemple 390. Exemple de l'opcode *prealloc*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o prealloc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Pre-allocate memory for five instances of Instrument #1.
prealloc 1, 5
```

```
; Instrument #1
instr 1
; Generate a waveform, get the cycles per second from the 4th p-field.
a1 oscil 6500, p4, 1
out a1
endin

</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play five instances of Instrument #1 for one second.
; Note that 4th p-field contains cycles per second.
i 1 0 1 220
i 1 0 1 440
i 1 0 1 880
i 1 0 1 1320
i 1 0 1 1760
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*cpuprc, maxalloc*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# prepiano

prepiano — Crée un son similaire à celui d'une corde de piano préparé à la manière Cage.

## Description

La sortie audio est un son similaire à celui d'une corde de piano préparé avec des gommes et des pièces de monnaie. La méthode utilise un modèle physique développé pour la résolution des équations différentielles partielles.

## Syntaxe

```
ares prepiano ifreq, iNS, iD, iK, \  
    iT30, iB, kbcl, kbcr, imass, ifreq, iinit, ipos, ivel, isfreq, \  
    isspread[, irattles, irubbers]  
  
al,ar prepiano ifreq, iNS, iD, iK, \  
    iT30, iB, kbcl, kbcr, imass, ifreq, iinit, ipos, ivel, isfreq, \  
    isspread[, irattles, irubbers]
```

## Initialisation

*ifreq* -- la fréquence de base de la corde.

*iNS* -- le nombre de cordes impliquées. Dans un vrai piano on trouve 1, 2 ou 3 cordes dans les différentes plages de fréquence.

*iD* -- l'importance du désaccord de chaque corde, hormis la première, par rapport à la fréquence principale ; mesuré en cents.

*iK* -- paramètre de raideur, sans dimension.

*iT30* -- durée de chute de 30 db en secondes.

*ib* -- paramètre de perte en haute-fréquence (à garder petit).

*imass* -- la masse du marteau.

*ifreq* -- la fréquence de vibration naturelle du marteau.

*iinit* -- la position initiale du marteau.

*ipos* -- position de la frappe sur la corde.

*ivel* -- vitesse normalisée de la frappe.

*isfreq* -- fréquence de balayage du point de lecture.

*isspread* -- dispersion de la fréquence de balayage.

*irattles* -- numéro de la table donnant les positions de la ou des pièces de monnaie.

*irubbers* -- numéro de la table donnant les positions de la ou des gommes.

Les tables des pièces de monnaie et des gommes sont des collections de quatre valeurs précédées par un

compte. Dans le cas d'une pièce de monnaie, les quatre valeurs sont la position, le rapport de densité entre la pièce de monnaie et la corde, la fréquence de la pièce de monnaie et sa longueur verticale. Pour la gomme, les quatre valeurs sont la position, le rapport de densité entre la gomme et la corde, la fréquence de la gomme et le paramètre de perte.

## Exécution

Une note est jouée sur une corde de piano avec les arguments suivants.

*kbcL* -- Condition aux limites à l'extrémité gauche de la corde (1 fixée, 2 pivotante, 3 libre).

*kbcR* -- Condition aux limites à l'extrémité droite de la corde (1 fixée, 2 pivotante, 3 libre).

Il faut noter que le changement des conditions aux limites durant l'exécution peut produire des bruits parasites et que cette possibilité n'est fournie qu'à titre expérimental.

## Exemples

Voici en exemple de l'opcode *prepiano*. Il utilise le fichier *prepiano.csd* [examples/prepiano.csd].

### Exemple 391. Exemple de l'opcode *prepiano*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o prepiano.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2;

; Instrument #1.
instr 1
;;      fund NS detune stiffness decay loss (bndry) (hammer) scan prep
aa,ab prepiano 60, 3, 10, p4, 3, 0.002, 2, 2, 1, 5000, -0.01, p5, p6, 0, 0.1, 1, 2
outs aa*.75, ab*.75
endin
</CsInstruments>
<CsScore>
f1 0 8 2 1 0.6 10 100 0.001 ;; 1 rattle
f2 0 8 2 1 0.7 50 500 1000 ;; 1 rubber
i1 0.0 0.5 1 0.09 20
i1 0.5 . -1 0.09 40      ;; 1 -> skip initialisation
i1 1.0 . -1 0.09 60
i1 1.5 . -1 0.09 80
i1 2.0 1.8 -1 0.09 100
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits



Auteur : Stefan Bilbao  
Université d'Edimbourg, UK  
Auteur : John ffitch  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 5.05 de Csound

# print

print — Affiche les valeurs de variables de taux-i.

## Description

Ces unités affichent des valeurs d'initialisation de l'orchestre.

## Syntaxe

```
print iarg [, iarg1] [, iarg2] [...]
```

## Initialisation

*iarg*, *iarg2*, ... -- arguments de taux-i.

## Exécution

*print* -- affiche la valeur courante des arguments (ou des expressions) de taux-i *iarg* à chaque passe d'initialisation dans l'instrument.



### Note

L'opcode *print* tronque des positions décimales et peut ainsi ne pas montrer la valeur complète. La précision de Csound varie selon la *version* float (32 bit) ou double (64 bit), car la plupart des calculs internes utilisent un de ces formats. Si l'on désire une sortie console avec plus de résolution, on peut essayer *printf*.

## Exemples

Voici un exemple de l'opcode *print*. Il utilise le fichier *print.csd* [examples/print.csd].

### Exemple 392. Exemple de l'opcode *print*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o print.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Print the fourth p-field.
  print p4
endin

</CsInstruments>
<CsScore>

  ; p4 = value to be printed.
  ; Play Instrument #1 for one second, p4 = 50.375.
  i 1 0 1 50.375
  ; Play Instrument #1 for one second, p4 = 300.
  i 1 1 1 300
  ; Play Instrument #1 for one second, p4 = -999.
  i 1 2 1 -999
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
instr 1:  p4 = 50.375
instr 1:  p4 = 300.000
instr 1:  p4 = -999.000
```

## Voir Aussi

*disppfft, display, printk, printk2, printks, printf and prints*

## Crédits

Exemple écrit par Kevin Conder.

# printf

printf — Sortie formatée à la façon printf.

## Description

**printf** et **printf\_i** écrivent une sortie formatée à la manière de la fonction C *printf()*. **printf\_i** ne s'exécute qu'au taux-i, tandis que **printf** s'exécute à la fois à l'initialisation et pendant l'exécution de la note.

## Syntaxe

```
printf_i Sfmt, itrig, [iarg1[, iarg2[, ... ]]]
```

```
printf Sfmt, ktrig, [xarg1[, xarg2[, ... ]]]
```

## Initialisation

*Sfmt* -- chaîne de formatage ayant la même structure que dans *printf* et dans d'autres fonctions C similaires, sauf que les modificateurs de longueur (l, ll, h, etc.) ne sont pas supportés. Les indicateurs de conversion suivants sont permis :

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

*iarg1*, *iarg2*, ... -- arguments d'entrée à formater (30 au maximum). Les formats entiers tels que %d arrondissent les valeurs d'entrée à l'entier le plus proche.

*itrig* -- s'il est supérieur à zéro, l'opcode effectue l'affichage ; sinon c'est une opération nulle.

## Exécution

*ktrig* -- s'il est supérieur à zéro et différent de sa valeur lors du cycle de contrôle précédent, l'opcode effectue l'affichage demandé. La valeur précédente initiale est fixée à zéro.

*xarg1*, *xarg2*, ... -- arguments d'entrée à formater (30 au maximum). Les formats entiers tels que %d arrondissent les valeurs d'entrée à l'entier le plus proche. Noter que seuls les arguments de taux-k et de taux-i sont valides (pas d'affichage au taux-a)

## Crédits

Auteur : Istvan Varga  
2005

# printk

printk — Affiche une valeur de taux-k à intervalles définis.

## Description

Affiche une valeur de taux-k à intervalles définis.

## Syntaxe

```
printk itime, kval [, ispace]
```

## Initialisation

*itime* -- intervalle en secondes entre les impressions.

*ispace* (facultatif, 0 par défaut) -- nombre d'espaces à insérer avant l'impression. (0 par défaut, max : 130)

## Exécution

*kval* -- La valeur de taux-k à afficher.

*printk* imprime une valeur de taux-k à chaque cycle-k, à chaque seconde ou à intervalles définis. Le numéro d'instrument est d'abord imprimé, puis le temps absolu en secondes, ensuite un nombre donné d'espaces, enfin la valeur de *kval*. Le nombre variable d'espaces permet de répartir différentes valeurs sur l'écran, de manière plus visible.

Cet opcode peut être exécuté à chaque cycle-k de l'instrument auquel il appartient. Pour cela, il faut mettre *itime* à 0.

Si *itime* est différent de 0, l'opcode imprime sur le premier cycle-k lors de son appel, puis chaque fois qu'une durée *itime* s'est écoulée. Le temps commence à s'écouler à partir de l'initialisation de l'opcode, typiquement à l'initialisation de l'instrument.

## Exemples

Voici un exemple de l'opcode *printk*. Il utilise le fichier *printk.csd* [examples/printk.csd].

### Exemple 393. Exemple de l'opcode printk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o printk.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Change a value linearly from 0 to 100,
; over the period defined by p3.
kval line 0, p3, 100

; Print the value of kval, once per second.
printk 1, kval
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

i 1 time 0.00002: 0.00000
i 1 time 1.00002: 20.01084
i 1 time 2.00002: 40.02999
i 1 time 3.00002: 60.04914
i 1 time 4.00002: 79.93327

```

## Voir Aussi

*printk2* and *printks*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Exemple écrit par Kevin Conder.

Merci à Luis Jure pour avoir signalé une erreur concernant le paramètre *itime*.

# printk2

printk2 — Affiche une nouvelle valeur chaque fois qu'une variable de contrôle change.

## Description

Affiche une nouvelle valeur chaque fois qu'une variable de contrôle change.

## Syntaxe

```
printk2 kvar [, inumspaces]
```

## Initialisation

*inumspaces* (facultatif, 0 par défaut) -- nombre d'espaces imprimés avant la valeur de *kvar*

## Exécution

*kvar* -- signal à imprimer

Dérivé du *printk* de Robin Whittle, il affiche une nouvelle valeur de *kvar* chaque fois que *kvar* change. Utile pour surveiller les changements des contrôles MIDI lorsque l'on utilise des réglettes.



### Avertissement

Ne pas utiliser cet opcode avec des signaux de taux-k normaux variant continuellement, car cela pourrait bloquer l'ordinateur, le taux d'impression devenant trop rapide.

## Exemples

Voici un exemple de l'opcode printk2. Il utilise le fichier *printk2.csd* [examples/printk2.csd].

### Exemple 394. Exemple de l'opcode printk2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o printk2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1
```

```
; Instrument #1.
instr 1
; Change a value linearly from 0 to 10,
; over the period defined by p3.
kval1 line 0, p3, 10

; If kval1 is greater than or equal to 5,
; then kval=2, else kval=1.
kval2 = (kval1 >= 5 ? 2 : 1)

; Print the value of kval2 when it changes.
printk2 kval2
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
i1      1.00000
i1      2.00000
```

## Voir Aussi

*printk* and *printks*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.48 de Csound



# printks

printks — Imprime au taux-k avec une syntaxe à la printf().

## Description

Imprime au taux-k avec une syntaxe à la *printf()*.

## Syntaxe

```
printks "string", itime [, kval1] [, kval2] [...]
```

## Initialisation

*"string"* -- la chaîne de caractères à imprimer. Peut contenir jusqu'à 8192 caractères et doit être entre guillemets.

*itime* -- intervalle en secondes entre les impressions.

## Exécution

*kval1*, *kval2*, ... (facultatif) -- Les valeurs de taux-k à imprimer. Celles-ci sont spécifiées dans « *string* » au moyen des indicateurs de valeur du C standard (%f, %d, etc.) dans l'ordre donné.

A partir de la version 4.23 de Csound, on peut utiliser autant de variables *kval* que l'on veut. Dans les versions antérieures à la 4.23, on doit donner 4 et seulement 4 kvals (mettant 0 pour les kvals non utilisées).

*printks* affiche des nombres et du texte qui peuvent être des valeurs de taux-i ou de taux-k. *printks* est extrêmement flexible, et si on l'utilise avec des codes de positionnement du curseur, il peut servir à écrire des valeurs spécifiques à certaines positions de l'écran pendant l'exécution de Csound.

Un mode d'opération spécial permet à *printks* de convertir le paramètre d'entrée *kval1* en valeur comprise entre 0 et 255 et de l'utiliser comme le premier caractère à imprimer. Un programme Csound peut ainsi envoyer des caractères arbitraires à la console. Pour cela, il faut que le premier caractère de la chaîne soit un # éventuellement suivi de texte normal et d'indicateurs de format.

Cet opcode peut être exécuté à chaque cycle-k de l'instrument auquel il appartient. Pour cela, il faut mettre *itime* à 0.

Si *itime* est différent de 0, l'opcode imprime sur le premier cycle-k lors de son appel, puis chaque fois qu'une durée *itime* s'est écoulée. Le temps commence à s'écouler à partir de l'initialisation de l'opcode, typiquement à l'initialisation de l'instrument.

## Formatage de l'Impression

Tous les caractères de contrôle de *printf()* du langage C standard peuvent être utilisés. Par exemple, si *kval1* = 153.26789, voici quelques-unes des options de formatage habituelles :

1. %f imprime avec toute la précision : 153.26789

2. %5.2f imprime : 153.26
3. %d n'imprime que la partie entière : 153
4. %c traite *kvalI* comme le code ASCII d'un caractère.

En plus de tous les codes de *printf()*, *printks* supporte ces codes de caractère utiles :

Code printks	Code de Caractère
\\r, \\R, %r, or %R	retour chariot (\r)
\\n, \\N, %n, %N	caractère de nouvelle ligne (\n)
\\t, \\T, %t, or %T	tabulation (\t)
%!	point-virgule (;) C'est nécessaire car un « ; » est interprété comme un commentaire.
^	caractère d'échappement (0x1B)
^ ^	accent circonflexe (^)
~	ESC[ (escape+[ est la séquence d'échappement des consoles ANSI)
~~	tilde (~)

Pour plus d'information sur le formatage à la *printf()*, consulter une documentation sur le langage C.



## Note

Avant la version 4.23, seul le code de format %f était supporté.

## Exemples

Voici un exemple de l'opcode *printks*. Il utilise le fichier *printks.csd* [examples/printks.csd].

### Exemple 395. Exemple de l'opcode *printks*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o printks.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Change a value linearly from 0 to 100,
```

```

; over the period defined by p3.
kup line 0, p3, 100
; Change a value linearly from 30 to 10,
; over the period defined by p3.
kdown line 30, p3, 10

; Print the value of kup and kdown, once per second.
printks "kup = %f, kdown = %f\\n", 1, kup, kdown
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

kup = 0.000000, kdown = 30.000000
kup = 20.010843, kdown = 25.962524
kup = 40.029991, kdown = 21.925049
kup = 60.049141, kdown = 17.887573
kup = 79.933266, kdown = 13.872493

```

## Voir Aussi

*printk2* and *printk*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Exemple écrit par Kevin Conder.

Merci à Luis Jure pour avoir signalé une erreur concernant le paramètre *itime*.

Merci à Matt Ingalls pour la mise à jour de la documentation pour la version 4.23.

# prints

prints — Imprime au taux-i avec une syntaxe à la `printf()`.

## Description

Imprime au taux-i avec une syntaxe à la *printf()*.

## Syntaxe

```
prints "string" [, kval1] [, kval2] [...]
```

## Initialisation

*"string"* -- la chaîne de caractères à imprimer. Peut contenir jusqu'à 8192 caractères et doit être entre guillemets.

## Exécution

*kval1, kval2, ...* (optional) -- Les valeurs de taux-k à imprimer. Celles-ci sont spécifiées dans « *string* » au moyen des indicateurs de valeur du C standard (%f, %d, etc.) dans l'ordre donné. Mettre 0 pour celles qui ne sont pas utilisées.

*prints* est semblable à l'opcode *printks* sauf qu'il opère au taux-i plutôt qu'au taux-k. Pour plus d'information sur le formatage de la sortie, veuillez consulter la documentation de *printks*.

## Exemples

Voici un exemple de l'opcode *prints*. Il utilise le fichier *prints.csd* [examples/prints.csd].

### Exemple 396. Exemple de l'opcode prints.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o prints.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Init-time print.
```

```
prints "%2.3f\\t%!%!%!%!%!semicolons!\\n", 1234.56789
endin
```

```
</CsInstruments>
<CsScore>
```

```
/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play instrument #1.
i 1 0 0.004
```

```
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
1234.568      ;;;;;semicolons!
```

## Voir Aussi

*printks*

## Crédits

Auteur : Matt Ingalls  
Janvier 2003

# product

product — Multiplie n'importe quel nombre de signaux de taux-a.

## Description

Multiplie n'importe quel nombre de signaux de taux-a.

## Syntaxe

```
ares product asig1, asig2 [, asig3] [...]
```

## Exécution

*asig1, asig2, asig3, ...* -- signaux de taux-a à multiplier.

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Avril 1999

Nouveau dans la version 3.54 de Csound

# pset

*pset* — Définit et initialise des tableaux numériques au chargement de l'orchestre.

## Description

Définit et initialise des tableaux numériques au chargement de l'orchestre.

## Syntaxe

```
pset icon1 [, icon2] [...]
```

## Initialisation

*icon1, icon2, ...* -- valeurs de preset pour un instrument MIDI

*pset* (facultatif) définit et initialise des tableaux numériques au chargement de l'orchestre. On peut l'utiliser comme instruction dans l'en-tête de l'orchestre (c'est-à-dire dans l'instrument 0) ou dans un instrument. Lorsqu'il est défini dans un instrument, il ne fait pas partie de ses opérations des périodes d'initialisation ou d'exécution, et une seule de ces instructions est autorisée par instrument. Ces valeurs sont disponibles comme valeurs d'initialisation par défaut. Quand un instrument est déclenché à partir du MIDI, il ne reçoit que p1 et p2 de l'évènement, alors que p3, p4, etc proviennent des valeurs définies dans le preset.

## Exemples

L'exemple ci-dessous illustre l'utilisation de *pset* dans un instrument.

```
instr 1  
  pset 0,0,3,4,5,6 ; pfield substitutes  
  a1 oscil 10000, 440, p6
```

## Voir Aussi

*strset*

# ptrack

ptrack — Tracks the pitch of a signal.

## Description

*ptrack* takes an input signal, splits it into *ihopsize* blocks and using a STFT method, extracts an estimated pitch for its fundamental frequency as well as estimating the total amplitude of the signal in dB, relative to full-scale (0dB). The method implies an analysis window size of  $2 \times ihopsize$  samples (overlapping by 1/2 window), which has to be a power-of-two, between 128 and 8192 (hopsizes between 64 and 4096). Smaller windows will give better time precision, but worse frequency accuracy (esp. in low fundamentals). This opcode is based on an original algorithm by M. Puckette.

## Syntax

```
kcps, kamp ptrack asig, ihopsize[,ipeaks]
```

## Initialization

*ihopsize* -- size of the analysis 'hop', in samples, required to be power-of-two (min 64, max 4096). This is the period between measurements.

*ipeaks, ihi* -- number of spectral peaks to use in the analysis, defaults to 20 (optional)

## Performance

*kcps* -- estimated pitch in Hz.

*kamp* -- estimated amplitude in dB relative to full-scale (0dB) (ie. always  $\leq 0$ ).

*ptrack* analyzes the input signal, *asig*, to give a pitch/amplitude pair of outputs, for the fundamental of a monophonic signal. The output is updated every  $sr/ihopsize$  seconds.

## Examples

Here is an example of the *ptrack* opcode. This example uses the files *ptrack.csd* [examples/ptrack.csd].

### Exemple 397. Example of the ptrack opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No display
-odac          -iadc     -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>
sr= 44100
ksmps = 16
nchnls= 1

;Example by Victor Lazzarini 2007
```



```
instr 1
  al inch 1          ; take an input signal
  kf,ka ptrack al, 512 ; pitch track with winsize=1024
  kcps port kf, 0.01  ; smooth freq
  kamp port ka, 0.01  ; smooth amp

  ; drive an oscillator
  aout oscili ampdb(kamp)*0dbfs, kcps, 1

  out aout
endin

</CsInstruments>
<CsScore>
il 0 3600
e
</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Victor Lazzarini  
NUI, Maynooth.  
Maynooth, Ireland  
March, 2007

New in Csound version 5.05

# puts

puts — Print a string constant or variable

## Description

puts prints a string with an optional newline at the end whenever the trigger signal is positive and changes.

## Syntax

```
puts Sstr, ktrig[, inonl]
```

## Initialization

*Sstr* -- string to be printed

*inonl* (optional, defaults to 0) -- if non-zero, disables the default printing of a newline character at the end of the string

## Performance

*ktrig* -- trigger signal, should be valid at i-time. The string is printed at initialization time if ktrig is positive, and at performance time whenever ktrig is both positive and different from the previous value. Use a constant value of 1 to print once at note initialization.

## Credits

Author: Istvan Varga  
2005

# push

push — Pushes a value into the global stack.

## Description

Pushes a value into the global stack.

## Syntax

```
push  xval1, [xval2, ... , xval31]
```

```
push  ival1, [ival2, ... , ival31]
```

## Initialization

*ival1 ... ival31* - values to be pushed into the stack.

## Performance

*xval1 ... xval31* - values to be pushed into the stack.

The given values are pushed into the global stack as a bundle. The global stack works in LIFO order: after multiple *push* calls, *pop* should be used in reverse order.

Each *push* or *pop* operation can work on a "bundle" of multiple variables. When using *pop*, the number, type, and order of items must match those used by the corresponding *push*. That is, after a 'push Sfoo, ibar', you must call something like 'pop Sbar, ifoo', and not e.g. two separate 'pop' statements.

*push* and *pop* opcodes can take variables of any type (i-, k-, a- and strings). Use of any combination of i, k, a, and S types is allowed. Variables of type 'a' and 'k' are passed at performance time only, while 'i' and 'S' are passed at init time only.

push/pop for a, k, i, and S types copy data by value. By contrast, *push\_f* only pushes a "reference" to the f-signal, and then the corresponding *pop\_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push\_f* before *pop\_f* is called is not recommended, and if the instrument instance owning the variable that was passed by *push\_f* is deactivated before *pop\_f* is called, undefined behavior may occur.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

## See also

*stack*, *pop*, *pop\_f* and *push\_f*.

## Credits

By: Istvan Varga.

2006

# push\_f

`push_f` — Pushes an f-sig frame into the global stack.

## Description

Pushes an f-sig frame into the global stack.

## Syntax

`push_f` *fsig*

## Performance

*fsig* - f-signal to be pushed into the stack.

The values are pushed into the global stack. The global stack works in LIFO order: after multiple *push\_f* calls, *pop\_f* should be used in reverse order.

push/pop for a, k, i, and S types copy data by value. By contrast, *push\_f* only pushes a "reference" to the f-signal, and then the corresponding *pop\_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push\_f* before *pop\_f* is called is not recommended, and if the instrument instance owning the variable that was passed by *push\_f* is deactivated before *pop\_f* is called, undefined behavior may occur.

*pop\_f* and *push\_f* can only take a single argument, and the data is passed both at init and performance time.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

## See also

*stack*, *push*, *pop* and *pop\_f*.

## Credits

By: Istvan Varga.

2006

# pvadd

pvadd — Reads from a *pvoc* file and uses the data to perform additive synthesis.

## Description

*pvadd* reads from a *pvoc* file and uses the data to perform additive synthesis using an internal array of interpolating oscillators. The user supplies the wave table (usually one period of a sine wave), and can choose which analysis bins will be used in the re-synthesis.

## Syntax

```
ares pvadd ktmpnt, kfmmod, ifilcod, ifn, ibins [, ibinoffset] \  
      [, ibinincr] [, iextractmode] [, ifreqlim] [, igatefn]
```

## Initialization

*ifilcod* -- integer or character-string denoting a control-file derived from *pvanal* analysis of an audio signal. An integer denotes the suffix of a file *pvoc.m*; a character-string (in double quotes) gives a file-name, optionally a full pathname. If not fullpath, the file is sought first in the current directory, then in the one given by the environment variable *SADIR* (if defined). *pvoc* control files contain data organized for fft resynthesis. Memory usage depends on the size of the files involved, which are read and held entirely in memory during computation but are shared by multiple calls (see also *lpread*).

*ifn* -- table number of a stored function containing a sine wave.

*ibins* -- number of bins that will be used in the resynthesis (each bin counts as one oscillator in the re-synthesis)

*ibinoffset* (optional) -- is the first bin used (it is optional and defaults to 0).

*ibinincr* (optional) -- sets an increment by which *pvadd* counts up from *ibinoffset* for *ibins* components in the re-synthesis (see below for a further explanation).

*iextractmode* (optional) -- determines if spectral extraction will be carried out and if so whether components that have changes in frequency below *ifreqlim* or above *ifreqlim* will be discarded. A value for *iextractmode* of 1 will cause *pvadd* to synthesize only those components where the frequency difference between analysis frames is greater than *ifreqlim*. A value of 2 for *iextractmode* will cause *pvadd* to synthesize only those components where the frequency difference between frames is less than *ifreqlim*. The default values for *iextractmode* and *ifreqlim* are 0, in which case a simple resynthesis will be done. See examples below.

*igatefn* (optional) -- is the number of a stored function which will be applied to the amplitudes of the analysis bins before resynthesis takes place. If *igatefn* is greater than 0 the amplitudes of each bin will be scaled by *igatefn* through a simple mapping process. First, the amplitudes of all of the bins in all of the frames in the entire analysis file are compared to determine the maximum amplitude value. This value is then used create normalized amplitudes as indices into the stored function *igatefn*. The maximum amplitude will map to the last point in the function. An amplitude of 0 will map to the first point in the function. Values between 0 and 1 will map accordingly to points along the function table. This will be made clearer in the examples below.

## Performance

*ktimpnt* and *kfmod* are used in the same way as in *pvoc*.

## Examples

```
ptime line 0, p3, p3
```

```
asig pvadd ptime, 1, « oboe.pvoc », 1, 100, 2
```

In the above, *ibins* is 100 and *ibinoffset* is 2. Using these settings the resynthesis will contain 100 components beginning with bin #2 (bins are counted starting with 0). That is, resynthesis will be done using bins 2-101 inclusive. It is usually a good idea to begin with bin 1 or 2 since the 0th and often 1st bin have data that is neither necessary nor even helpful for creating good clean resynthesis.

```
ptime line 0, p3, p3
asig pvadd ptime, 1, « oboe.pvoc », 1, 100, 2, 2
```

The above is the same as the previous example with the addition of the value 2 used for the optional *ibinincr* argument. This result will still result in 100 components in the resynthesis, but *pvadd* will count through the bins by 2 instead of by 1. It will use bins 2, 4, 6, 8, 10, and so on. For *ibins*=10, *ibinoffset*=10, and *ibinincr*=10, *pvadd* would use bins 10, 20, 30, 40, up to and including 100.

Below is an example using spectral extraction. In this example *iextractmode* is one and *ifreqlim* is 9. This will cause *pvadd* to synthesize only those bins where the frequency deviation, averaged over 6 frames, is greater than 9.

```
ptime line 0, p3, p3
asig pvadd ptime, 1, « oboe.pvoc », 1, 100, 2, 2, 1, 9
```

If *iextractmode* were 2 in the above, then only those bins with an average frequency deviation of less than 9 would be synthesized. If tuned correctly, this technique can be used to separate the pitched parts of the spectrum from the noisy parts. In practice this depends greatly on the type of sound, the quality of the recording and digitization, and also on the analysis window size and frame increment.

Next is an example using amplitude gating. The last 2 in the argument list stands for *f2* in the score.

```
asig pvadd ptime, 1, « oboe.pvoc », 1, 100, 2, 2, 0, 0, 2
```

Suppose the score for the above were to contain:

```
f2 0 512 7 0 256 1 256 1
```

Then those bins with amplitudes of 50% of the maximum or greater would be left unchanged, while those with amplitudes less than 50% of the maximum would be scaled down. In this case the lower the amplitude the more severe the scaling down would be. But suppose the score contains:

```
f2 0 512 5 1 512 .001
```

In this case lower amplitudes will be left unchanged and greater ones will be scaled down, turning the sound « upside-down » in terms of the amplitude spectrum! Functions can be arbitrarily complex. Just remember that the normalized amplitude values of the analysis are themselves the indices into the function.

Finally, both spectral extraction and amplitude gating can be used together. The example below will synthesize only those components that with a frequency deviation of less than 5Hz per frame and it will scale the amplitudes according to F2.

```
asig pvadd ktime, 1, « oboe.pvoc », 1, 100, 1, 1, 2, 5, 2
```



## USEFUL HINTS

By using several *pvadd* units together, one can gradually fade in different parts of the re-synthesis, creating various « filtering » effects. The author uses *pvadd* to synthesize one bin at a time to have control over each separate component of the re-synthesis.

If any combination of *ibins*, *ibinoffset*, and *ibinincr*, creates a situation where *pvadd* is asked to use a bin number greater than the number of bins in the analysis, it will just use all of the available bins, and give no complaint. So to use every bin just make *ibins* a big number (ie. 2000).

Expect to have to scale up the amplitudes by factors of 10-100, by the way.

## Credits

Author: Richard Karpen  
Seattle, WA USA  
1998

New in Csound version 3.48, additional arguments version 3.56

# pvbufread

pvbufread — Reads from a phase vocoder analysis file and makes the retrieved data available.

## Description

*pvbufread* reads from a *pvoc* file and makes the retrieved data available to any following *pvinterp* and *pvcross* units that appear in an instrument before a subsequent *pvbufread* (just as *lpread* and *lpreson* work together). The data is passed internally and the unit has no output of its own.

## Syntax

```
pvbufread ktimepnt, ifile
```

## Initialization

*ifile* -- the *pvoc* number (n in *pvoc.n*) or the name in quotes of the analysis file made using *pvanal*. (See *pvoc*.)

## Performance

*ktimepnt* -- the passage of time, in seconds, through this file. *ktimepnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

## Examples

The example below shows an example using *pvbufread* with *pvinterp* to interpolate between the sound of an oboe and the sound of a clarinet. The value of *kinterp* returned by a *linseg* is used to determine the timing of the transitions between the two sounds. The interpolation of frequencies and amplitudes are controlled by the same factor in this example, but for other effects it might be interesting to not have them synchronized in this way. In this example the sound will begin as a clarinet, transform into the oboe and then return again to the clarinet sound. The value of *kfreqscale2* is 1.065 because the oboe in this case is a semitone higher in pitch than the clarinet and this brings them approximately to the same pitch. The value of *kampscale2* is .75 because the analyzed clarinet was somewhat louder than the analyzed oboe. The setting of these two parameters make the transition quite smooth in this case, but such adjustments are by no means necessary or even advocated.

```
ktime1 line      0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2 line      0, p3, 4.5 ; used as index in the "clar.pvoc" file
kinterp linseg   1, p3*.15, 1, p3*.35, 0, p3*.25, 0, p3*.15, 1, p3*.1, 1
pvbufread ktime1, "oboe.pvoc"
apv pvinterp ktime2,1,"clar.pvoc",1,1.065,1,.75,1-kinterp,1-kinterp
```

Below is an example using *pvbufread* with *pvcross*. In this example the amplitudes used in the resynthesis gradually change from those of the oboe to those of the clarinet. The frequencies, of course, remain those of the clarinet throughout the process since *pvcross* does not use the frequency data from the file read by *pvbufread*.



```
ptime1  line    0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ptime2  line    0, p3, 4.5 ; used as index in the "clar.pvoc" file
kcross  expon    .001, p3, 1
        pvbufread ptime1, "oboe.pvoc"
apv      pvcross  ptime2, 1, "clar.pvoc", 1-kcross, kcross
```

## See Also

*pvcross, pvinterp, pvread, tableseg, tablexseg*

## Credits

Author: Richard Karpen  
Seattle, WA USA  
1997

# pvcross

pvcross — Applies the amplitudes from one phase vocoder analysis file to the data from a second file.

## Description

*pvcross* applies the amplitudes from one phase vocoder analysis file to the data from a second file and then performs the resynthesis. The data is passed, as described above, from a previously called *pvburead* unit. The two k-rate amplitude arguments are used to scale the amplitudes of each files separately before they are added together and used in the resynthesis (see below for further explanation). The frequencies of the first file are not used at all in this process. This unit simply allows for cross-synthesis through the application of the amplitudes of the spectra of one signal to the frequencies of a second signal. Unlike *pvinterp*, *pvcross* does allow for the use of the *ispecwp* as in *pvoc* and *vpvoc*.

## Syntax

```
ares pvcross ktmpnt, kfmod, ifile, kampscale1, kampscale2 [, ispecwp]
```

## Initialization

*ifile* -- the *pvoc* number (n in *pvoc.n*) or the name in quotes of the analysis file made using *pvanal*. (See *pvoc*.)

*ispecwp* (optional, default=0) -- if non-zero, attempts to preserve the spectral envelope while its frequency content is varied by *kfmod*. The default value is zero.

## Performance

*ktmpnt* -- the passage of time, in seconds, through this file. *ktmpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

*kfmod* -- a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave.

*kampscale1*, *kampscale2* -- used to scale the amplitudes stored in each frame of the phase vocoder analysis file. *kampscale1* scale the amplitudes of the data from the file read by the previously called *pvburead*. *kampscale2* scale the amplitudes of the file named by *ifile*.

By using these arguments, it is possible to adjust these values before applying the interpolation. For example, if file1 is much louder than file2, it might be desirable to scale down the amplitudes of file1 or scale up those of file2 before interpolating. Likewise one can adjust the frequencies of each to bring them more in accord with one another (or just the opposite, of course!) before the interpolation is performed.

## Examples

Below is an example using *pvburead* with *pvcross*. In this example the amplitudes used in the resynthesis gradually change from those of the oboe to those of the clarinet. The frequencies, of course, remain those of the clarinet throughout the process since *pvcross* does not use the frequency data from the file read by *pvburead*.

```
ptime1  line    0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ptime2  line    0, p3, 4.5 ; used as index in the "clar.pvoc" file
kcross  expon    .001, p3, 1
        pvbufread ptime1, "oboe.pvoc"
apv      pvcross  ptime2, 1, "clar.pvoc", 1-kcross, kcross
```

## See Also

*pvbufread, pvinterp, pvread, tableseg, tablexseg*

## Credits

Author: Richard Karpen  
Seattle, Wash  
1997

New in version 3.44

# pvinterp

pvinterp — Interpolates between the amplitudes and frequencies of two phase vocoder analysis files.

## Description

*pvinterp* interpolates between the amplitudes and frequencies, on a bin by bin basis, of two phase vocoder analysis files (one from a previously called *pdbufread* unit and the other from within its own argument list), allowing for user defined transitions between analyzed sounds. It also allows for general scaling of the amplitudes and frequencies of each file separately before the interpolated values are calculated and sent to the resynthesis routines. The *kfmod* argument in *pvinterp* performs its frequency scaling on the frequency values after their derivation from the separate scaling and subsequent interpolation is performed so that this acts as an overall scaling value of the new frequency components.

## Syntax

```
ares pvinterp ktmpnt, kfmod, ifile, kfreqscale1, kfreqscale2, \  
      kampscale1, kampscale2, kfreqinterp, kampinterp
```

## Initialization

*ifile* -- the *pvoc* number (n in *pvoc.n*) or the name in quotes of the analysis file made using *pvanal*. (See *pvoc*.)

## Performance

*ktmpnt* -- the passage of time, in seconds, through this file. *ktmpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

*kfmod* -- a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave.

*kfreqscale1*, *kfreqscale2*, *kampscale1*, *kampscale2* -- used in *pvinterp* to scale the frequencies and amplitudes stored in each frame of the phase vocoder analysis file. *kfreqscale1* and *kampscale1* scale the frequencies and amplitudes of the data from the file read by the previously called *pdbufread* (this data is passed internally to the *pvinterp* unit). *kfreqscale2* and *kampscale2* scale the frequencies and amplitudes of the file named by *ifile* in the *pvinterp* argument list and read within the *pvinterp* unit.

By using these arguments, it is possible to adjust these values before applying the interpolation. For example, if file1 is much louder than file2, it might be desirable to scale down the amplitudes of file1 or scale up those of file2 before interpolating. Likewise one can adjust the frequencies of each to bring them more in accord with one another (or just the opposite, of course!) before the interpolation is performed.

*kfreqinterp*, *kampinterp* -- used in *pvinterp*, determine the interpolation distance between the values of one phase vocoder file and the values of a second file. When the value of *kfreqinterp* is 1, the frequency values will be entirely those from the first file (read by the *pdbufread*), post scaling by the *kfreqscale1* argument. When the value of *kfreqinterp* is 0 the frequency values will be those of the second file (read by the *pvinterp* unit itself), post scaling by *kfreqscale2*. When *kfreqinterp* is between 0 and 1 the frequency values will be calculated, on a bin, by bin basis, as the percentage between each pair of frequencies (in other words, *kfreqinterp*=.5 will cause the frequencies values to be half way between the values in the set of data from the first file and the set of data from the second file).

*kampinterp* works in the same way upon the amplitudes of the two files. Since these are k-rate arguments, the percentages can change over time making it possible to create many kinds of transitions between sounds.

## Examples

The example below shows an example using *pvbufread* with *pvinterp* to interpolate between the sound of an oboe and the sound of a clarinet. The value of *kinterp* returned by a *linseg* is used to determine the timing of the transitions between the two sounds. The interpolation of frequencies and amplitudes are controlled by the same factor in this example, but for other effects it might be interesting to not have them synchronized in this way. In this example the sound will begin as a clarinet, transform into the oboe and then return again to the clarinet sound. The value of *kfreqscale2* is 1.065 because the oboe in this case is a semitone higher in pitch than the clarinet and this brings them approximately to the same pitch. The value of *kampscale2* is .75 because the analyzed clarinet was somewhat louder than the analyzed oboe. The setting of these two parameters make the transition quite smooth in this case, but such adjustments are by no means necessary or even advocated.

```

ktime1 line      0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2 line      0, p3, 4.5 ; used as index in the "clar.pvoc" file
kinterp linseg    1, p3*.15, 1, p3*.35, 0, p3*.25, 0, p3*.15, 1, p3*.1, 1
          pvbufread ktime1, "oboe.pvoc"
apv       pvinterp ktime2,1,"clar.pvoc",1,1.065,1,.75,1-kinterp,1-kinterp

```

## See Also

*pvbufread*, *pvcross*, *pvread*, *tableseg*, *tablexseg*

## Credits

Author: Richard Karpen  
 Seattle, Wash  
 1997

# pvoc

pvoc — Implements signal reconstruction using an fft-based phase vocoder.

## Description

Implements signal reconstruction using an fft-based phase vocoder.

## Syntax

```
ares pvoc ktmpnt, kfmod, ifilcod [, ispecwp] [, iextractmode] \  
      [, ifreqlim] [, igatefn]
```

## Initialization

*ifilcod* -- integer or character-string denoting a control-file derived from analysis of an audio signal. An integer denotes the suffix of a file *pvoc.m*; a character-string (in double quotes) gives a filename, optionally a full pathname. If not fullpath, the file is sought first in the current directory, then in the one given by the environment variable *SADIR* (if defined). *pvoc* control contains breakpoint amplitude and frequency envelope values organized for fft resynthesis. Memory usage depends on the size of the files involved, which are read and held entirely in memory during computation but are shared by multiple calls (see also *lpread*).

*ispecwp* (optional) -- if non-zero, attempts to preserve the spectral envelope while its frequency content is varied by *kfmod*. The default value is zero.

*extractmode* (optional) -- determines if spectral extraction will be carried out and if so whether components that have changes in frequency below *ifreqlim* or above *ifreqlim* will be discarded. A value for *extractmode* of 1 will cause *pvadd* to synthesize only those components where the frequency difference between analysis frames is greater than *ifreqlim*. A value of 2 for *extractmode* will cause *pvadd* to synthesize only those components where the frequency difference between frames is less than *ifreqlim*. The default values for *extractmode* and *ifreqlim* are 0, in which case a simple resynthesis will be done. See examples under *pvadd* for how to use spectral extraction.

*igatefn* (optional) -- the number of a stored function which will be applied to the amplitudes of the analysis bins before resynthesis takes place. If *igatefn* is greater than 0 the amplitudes of each bin will be scaled by *igatefn* through a simple mapping process. First, the amplitudes of all of the bins in all of the frames in the entire analysis file are compared to determine the maximum amplitude value. This value is then used create normalized amplitudes as indices into the stored function *igatefn*. The maximum amplitude will map to the last point in the function. An amplitude of 0 will map to the first point in the function. Values between 0 and 1 will map accordingly to points along the function table. See examples under *pvadd* for how to use amplitude gating.

## Performance

*ktmpnt* -- The passage of time, in seconds, through the analysis file. *ktmpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

*kfmod* -- a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave.

*pvoc* implements signal reconstruction using an fft-based phase vocoder. The control data stems from a

precomputed analysis file with a known frame rate.

This implementation of *pvoc* was originally written by Dan Ellis. It is based in part on the system of Mark Dolson, but the pre-analysis concept is new. The spectral extraction and amplitude gating (new in Csound version 3.56) were added by Richard Karpen based on functions in SoundHack by Tom Erbe.

## See Also

*vpvoc*, *PVANAL*.

## Credits

Authors: Dan Ellis and Richard Karpen  
Seattle, Wash  
1997

# pvread

**pvread** — Reads from a phase vocoder analysis file and returns the frequency and amplitude from a single analysis channel or bin.

## Description

*pvread* reads from a *pvoc* file and returns the frequency and amplitude from a single analysis channel or bin. The returned values can be used anywhere else in the Csound instrument. For example, one can use them as arguments to an oscillator to synthesize a single component from an analyzed signal or a bank of *pvreads* can be used to resynthesize the analyzed sound using additive synthesis by passing the frequency and magnitude values to a bank of oscillators.

## Syntax

```
kfreq, kamp pvread ktime, ifile, ibin
```

## Initialization

*ifile* -- the *pvoc* number (n in *pvoc.n*) or the name in quotes of the analysis file made using *pvanal*. (See *pvoc*.)

*ibin* -- the number of the analysis channel from which to return frequency in Hz and magnitude.

## Performance

*kfreq*, *kamp* -- outputs of the *pvread* unit. These values, retrieved from a phase vocoder analysis file, represent the values of frequency and amplitude from a single analysis channel specified in the *ibin* argument. Interpolation between analysis frames is performed at k-rate resolution and dependent of course upon the rate and direction of *ktime*.

*ktime* -- the passage of time, in seconds, through this file. *ktime* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

## Examples

The example below shows the use *pvread* to synthesize a single component from a phase vocoder analysis file. It should be noted that the *kfreq* and *kamp* outputs can be used for any kind of synthesis, filtering, processing, and so on.

```
ktime      line      0, p3, 3
kfreq, kamp  pvread  ktime, "pvoc.file", 7 ; read
                                     ;data from 7th analysis bin.
asig        oscili   kamp, kfreq, 1      ; function 1
                                     ;is a stored sine
```

## See Also



*pvbufread, pvcross, pvinterp, tableseg, tablexseg*

## Credits

Author: Richard Karpen  
Seattle, Wash  
1997

New in version 3.44

# pvsadsyn

pvsadsyn — Resynthesize using a fast oscillator-bank.

## Description

Resynthesize using a fast oscillator-bank.

## Syntax

```
ares pvsadsyn fsrc, inoscs, kfmod [, ibinoffset] [, ibinincr] [, iinit]
```

## Initialization

*inoscs* -- The number of analysis bins to synthesise. Cannot be larger than the size of *fsrc* (see *pvsinfo*), e.g. as created by *pvsanal*. Processing time is directly proportional to *inoscs*.

*ibinoffset* (optional, default=0) -- The first (lowest) bin to resynthesise, counting from 0 (default = 0).

*ibinincr* (optional) -- Starting from bin *ibinoffset*, resynthesize bins *ibinincr* apart.

*iinit* (optional) -- Skip reinitialization. This is not currently implemented for any of these opcodes, and it remains to be seen if it is even practical.

## Performance

*kfmod* -- Scale all frequencies by factor *kfmod*. 1.0 = no change, 2 = up one octave.

*pvsadsyn* is experimental, and implements the oscillator bank using a fast direct calculation method, rather than a lookup table. This takes advantage of the fact, empirically arrived at, that for the analysis rates generally used, (and presuming analysis using *pvsanal*, where frequencies in a bin change only slightly between frames) it is not necessary to interpolate frequencies between frames, only amplitudes. Accurate resynthesis is often contingent on the use of *pvsanal* with *iwinsize* = *ifftsize*\*2.

This opcode is the most likely to change, or be much extended, according to feedback and advice from users. It is likely that a full interpolating table-based method will be added, via a further optional *iarg*. The parameter list to *pvsadsyn* mimics that for *pvsadd*, but excludes spectral extraction.

## Examples

Here is an example of the *pvsadsyn* opcode. It uses the file *pvsadsyn.csd* [examples/pvsadsyn.csd].

### Exemple 398. Example of the pvsadsyn opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
```

```

-odac      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvadsyn.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

opcode FileToPvsBuf, iik, Siiii
;;writes an audio file at the first k-cycle to a fft-buffer (via pvsbuffer)
Sfile, ifftsize, ioverlap, iwinsize, iwinshape xin
ktimek      timeinstk
if ktimek == 1 then
  ilen      filelen Sfile
  kcycles =   ilen * kr; number of k-cycles to write the fft-buffer
  kcount     init      0
  loop:
    ain      soundin Sfile
    fftin    pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape
    ibuf, ktim pvsbuffer fftin, ilen + (ifftsize / sr)
    loop_lt kcount, 1, kcycles, loop
    xout      ibuf, ilen, ktim
  endif
endop

instr 1
  istretch =      p4; time stretching factor
  ifftsize =      1024
  ioverlap =      ifftsize / 4
  iwinsize =      ifftsize
  iwinshape =      1; von-Hann window
  ibuffer, ilen, k0      FileToPvsBuf "fox.wav", ifftsize, ioverlap, iwinsize, iwinshape
  p3      =      istretch * ilen; set p3 to the correct value
  ktmpnt      linseg      0, p3, ilen; time pointer
  fread      pvsbufread      ktmpnt, ibuffer; read the buffer
  aout      pvadsyn fread, 10, 1; resynthesis with the first 10 bins
  out      aout
endin

</CsInstruments>
<CsScore>
i 1 0 1 20
e
</CsScore>
</CsoundSynthesizer>

```

## See Also

*pvsanal, pvsynth, pvadsyn*

## Credits

Author: Richard Dobson  
August 2001

New in version 4.13

# pvsanal

pvsanal — Generate an fsig from a mono audio source ain, using phase vocoder overlap-add analysis.

## Description

Generate an fsig from a mono audio source ain, using phase vocoder overlap-add analysis.

## Syntax

```
fsig pvsanal ain, ifftsize, ioverlap, iwinsize, iwintype [, iformat] [, iinit]
```

## Initialization

*ifftsize* -- The FFT size in samples. Need not be a power of two (though these are especially efficient), but must be even. Odd numbers are rounded up internally. *ifftsize* determines the number of analysis bins in *fsig*, as  $\text{ifftsize}/2 + 1$ . For example, where *ifftsize* = 1024, *fsig* will contain 513 analysis bins, ordered linearly from the fundamental to Nyquist. The fundamental of analysis (which in principle gives the lowest resolvable frequency) is determined as  $\text{sr}/\text{ifftsize}$ . Thus, for the example just given and assuming  $\text{sr} = 44100$ , the fundamental of analysis is 43.07Hz. In practice, due to the phase-preserving nature of the phase vocoder, the frequency of any bin can deviate bilaterally, so that DC components are recorded. Given a strongly pitched signal, frequencies in adjacent bins can bunch very closely together, around partials in the source, and the lowest bins may even have negative frequencies.

As a rule, the only reason to use a non power-of-two value for *ifftsize* would be to match the known fundamental frequency of a strongly pitched source. Values with many small factors can be almost as efficient as power-of-two sizes; for example: 384, for a source pitched at around low A=110Hz.

*ioverlap* -- The distance in samples (« hop size ») between overlapping analysis frames. As a rule, this needs to be at least  $\text{ifftsize}/4$ , e.g. 256 for the example above. *ioverlap* determines the underlying analysis rate, as  $\text{sr}/\text{ioverlap}$ . *ioverlap* does not require to be a simple factor of *ifftsize*; for example a value of 160 would be legal. The choice of *ioverlap* may be dictated by the degree of pitch modification applied to the *fsig*, if any. As a rule of thumb, the more extreme the pitch shift, the higher the analysis rate needs to be, and hence the smaller the value for *ioverlap*. A higher analysis rate can also be advantageous with broadband transient sounds, such as drums (where a small analysis window gives less smearing, but more frequency-related errors).

Note that it is possible, and reasonable, to have distinct *fsigs* in an orchestra (even in the same instrument), running at different analysis rates. Interactions between such *fsigs* is currently unsupported, and the *fsig* assignment opcode does not allow copying between *fsigs* with different properties, even if the only difference is in *ioverlap*. However, this is not a closed issue, as it is possible in theory to achieve crude rate conversion (especially with regard to in-memory analysis files) in ways analogous to time-domain techniques.

*iwinsize* -- The size in samples of the analysis window filter (as set by *iwintype*). This must be at least *ifftsize*, and can usefully be larger. Though other proportions are permitted, it is recommended that *iwinsize* always be an integral multiple of *ifftsize*, e.g. 2048 for the example above. Internally, the analysis window (Hamming, von Hann) is multiplied by a sinc function, so that amplitudes are zero at the boundaries between frames. The larger analysis window size has been found to be especially important for oscillator bank resynthesis (e.g. using *pvsadsyn*), as it has the effect of increasing the frequency resolution of the analysis, and hence the accuracy of the resynthesis. As noted above, *iwinsize* determines the overall latency of the analysis/resynthesis system. In many cases, and especially in the absence of pitch modifications, it will be found that setting  $\text{iwinsize}=\text{ifftsize}$  works very well, and offers the lowest laten-

cy.

*iwintype* -- The shape of the analysis window. Currently only two choices are implemented:

- 0 = Hamming window
- 1 = von Hann window

Both are also supported by the PVOC-EX file format. The window type is stored as an internal attribute of the fsig, together with the other parameters (see *pvsinfo*). Other types may be implemented later on (e.g. the Kaiser window, also supported by PVOC-EX), though an obvious alternative is to enable windows to be defined via a function table. The main issue here is the constraint of f-tables to power-of-two sizes, so this method does not offer a complete solution. Most users will find the Hamming window meets all normal needs, and can be regarded as the default choice.

*iformat* -- (optional) The analysis format. Currently only one format is implemented by this opcode:

- 0 = amplitude + frequency

This is the classic phase vocoder format; easy to process, and a natural format for oscillator-bank resynthesis. It would be very easy (tempting, one might say) to treat an fsig frame not purely as a phase vocoder frame but as a generic additive synthesis frame. It is indeed possible to use an fsig this way, but it is important to bear in mind that the two are not, strictly speaking, directly equivalent.

Other important formats (supported by PVOC-EX) are:

- 1 = amplitude + phase
- 2 = complex (real + imaginary)

*iformat* is provided in case it proves useful later to add support for these other formats. Formats 0 and 1 are very closely related (as the phase is « wrapped » in both cases - it is a trivial matter to convert from one to the other), but the complex format might warrant a second explicit signal type (a « csig ») specifically for convolution-based processes, and other processes where the full complement of arithmetic operators may be useful.

*iinit* -- (optional) Skip reinitialization. This is not currently implemented for any of these opcodes, and it remains to be seen if it is even practical.



## Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

```
ain    in                ; live source
ffin   pvsanal   ain,1024,256,2048,0 ; analyze, using Hamming
ffout  pvsmask   ffin,1,0.75         ; apply eq from f-table
aout   pvsynth   ffout              ; and resynthesize
```

## Credits

Author: Richard Dobson  
August 2001

New in version 4.13

# pvsarp

pvsarp — Arpeggiate the spectral components of a streaming pv signal.

## Description

This opcode arpeggiates spectral components, by amplifying one bin and attenuating all the others around it. Used with an LFO it will provide a spectral arpeggiator similar to Trevor Wishart's CDP program specarp.

## Syntax

`fsig pvsarp fsigin, kbin, kdepth, kgain`

## Performance

*fsig* -- output pv stream

*fsigin* -- input pv stream

*kbin* -- target bin, normalised 0 - 1 (0Hz - Nyquist).

*kdepth* -- depth of attenuation of surrounding bins

*kgain* -- gain boost applied to target bin



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

Here is an example of the pvsarp opcode. It uses the file *pvsarp.csd* [examples/pvsarp.csd]

### Exemple 399. Example of the pvsarp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          ;; -d      RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvsarp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
ksmps = 300
nchnls = 1
0dbfs = 1

instr 1
  asig in ; get the signal in
  idepth = p4

  fsig pvsanal asig, 1024, 256, 1024, 1 ; analyse it
  kbin oscili 0.1, 0.5, 1 ; ftable 1 in the 0-1 range
  ftps pvsarp fsig, kbin+0.01, idepth, 2 ; arpeggiate it (range 220.5 - 2425.5)
  atps pvsynth ftps ; synthesise it

  out atps
endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ;sine wave
i 1 0 10 0.9
i 1 + 10 0.5
e

</CsScore>
</CsoundSynthesizer>
```

Here is another example of the pvsarp opcode. It uses the file *pvsarp2.csd* [examples/pvsarp2.csd]

## Exemple 400. Example of the pvsarp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in
-odac ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsarp2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
  ifftsize = 1024
  ioverlap = ifftsize / 4
  iwinsize = ifftsize
  iwinshape = 1; von-Hann window
  Sfile1 = "fox.wav"
  ain1 soundin Sfile1
  fftin pvsanal ain1, ifftsize, ioverlap, iwinsize, iwinshape
  ;make 3 independently moving accentuations in the spectrum
  kbin1 linseg 0.05, p3/2, .05, p3/2, .05
  farp1 pvsarp fftin, kbin1, .9, 10
  kbin2 linseg 0.075, p3/2, .1, p3/2, .075
  farp2 pvsarp fftin, kbin2, .9, 10
  kbin3 linseg 0.02, p3/2, .03, p3/2, .04
  farp3 pvsarp fftin, kbin3, .9, 10
  ;resynthesize and add them
  aout1 pvsynth farp1
  aout2 pvsynth farp2
  aout3 pvsynth farp3
  aout = aout1*.3 + aout2*.3 + aout3*.3
  out aout
```



```
endin
</CsInstruments>
<CsScore>
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal, pvsynth, pvsadsyn*

## Credits

Author: Victor Lazzarini  
April 2005

New plugin in version 5

April 2005.

# pvsbandp

pvsbandp — A band pass filter working in the spectral domain.

## Description

Filter the pvoc frames, passing bins whose frequency is within a band, and with linear interpolation for transitional bands.

## Syntax

```
fsig pvsbandp fsigin, xlowcut,  
xlowfull, xhighfull, xhighcut[, ktype]
```

## Performance

*f`sig`* -- output pv stream

*f`sigin`* -- input pv stream.

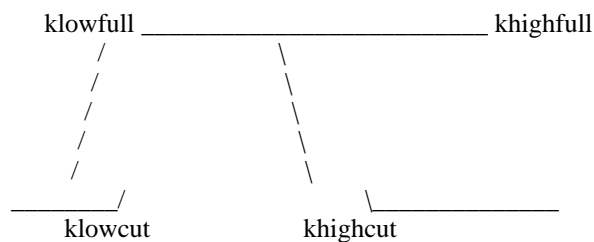
*x`lowcut`*, *x`lowfull`*, *x`highfull`*, *x`highcut`* -- define a trapezium shape for the band that is passed. The a-rate versions only apply to the sliding case.

*k`type`* -- specifies the shape of the transitional band. If at the default value of zero the shape is as below, with linear transition in amplitude. Other values yield and exponential shape:

$$(1 - \exp(r * \text{type})) / (1 - \exp(\text{type}))$$

This includes a linear dB shape when *k`type`* is  $\log(10)$  or about 2.30.

The opcode performs a band-pass filter with a spectral envelope shaped like



## Examples

Here is an example of the pvsbandp opcode. It uses the file *pvsbandp.csd* [examples/pvsbandp.csd].

### Exemple 401. Example of the pvsbandp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsbandp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
Sfile      =          "fox.wav"
klowcut = 100
klowfull = 200
khighfull = 1900
khighcut = 2000
ain        soundin Sfile
fftin      pvsanal ain, 1024, 256, 1024, 1; fft-analysis of the audio-signal
fftbp      pvsbandp fftin, klowcut, klowfull, khighfull, khighcut ; band pass
abp        pvsynth fftbp; resynthesis
           out      abp
endin

</CsInstruments>
<CsScore>
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal, pvsynth, pvsadsyn, pvsadsyn*

## Credits

Author: John ffitch  
December 2007

# pvsbandr

pvsbandr — A band reject filter working in the spectral domain.

## Description

Filter the pvoc frames, rejecting bins whose frequency is within a band, and with linear interpolation for transitional bands.

## Syntax

```
fsig pvsbandr fsigin, xlowcut,
      xlowfull, xhighfull, xhighcut[, ktype]
```

## Performance

*fsig* -- output pv stream

*fsigin* -- input pv stream.

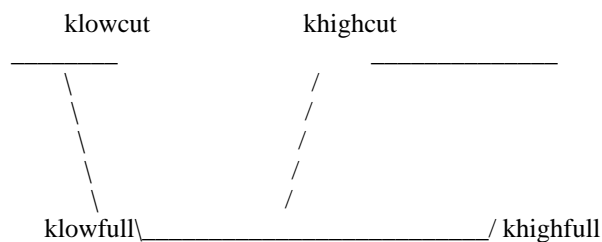
*xlowcut*, *xlowfull*, *xhighfull*, *xhighcut* -- define a trapezium shape for the band that is rejected. The a-rate versions only apply to the sliding case.

*ktype* -- specifies the shape of the transitional band. If at the default value of zero the shape is as below, with linear transition in amplitude. Other values give an exponential curve

$$(1 - \exp(-r \cdot \text{type})) / (1 - \exp(\text{type}))$$

This includes a linear dB shape when ktype is log(10) or about 2.30.

The opcode performs a band-reject filter with a spectral envelope shaped like



## Examples

Here is an example of the pvsbandr opcode. It uses the file *pvsbandr.csd* [examples/pvsbandr.csd].

### Exemple 402. Example of the pvsbandr opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsbandr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
Sfile      =          "fox.wav"
klowcut = 100
klowfull = 200
khighfull = 1900
khighcut = 2000
ain        soundin Sfile
fftin      pvsanal ain, 1024, 256, 1024, 1; fft-analysis of the audio-signal
fftbp      pvsbandr fftin, klowcut, klowfull, khighfull, khighcut ; band reject
abp        pvsynth fftbp; resynthesis
           out      abp
endin

</CsInstruments>
<CsScore>
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal, pvsynth, pvsadsyn, pvsadsyn*

## Credits

Author: John ffitch  
December 2007

# pvsbin

pvsbin — Obtain the amp and freq values off a PVS signal bin.

## Description

Obtain the amp and freq values off a PVS signal bin as k-rate variables.

## Syntax

```
kamp, kfr pvsbin fsig, kbin
```

## Performance

*kamp* -- bin amplitude

*kfr* -- bin frequency

*fsig* -- an input pv stream

*kbin* -- bin number

## Examples

Here is an example of the pvsbin opcode. It uses the file *pvsbin.csd* [examples/pvsbin.csd]. This example uses realtime input, but you can also use it for soundfile input.

### Exemple 403. Example of the pvsbin opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsbin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
ifftsize = 1024
iwtype = 1      /* cleaner with hanning window */

;al  soundin "input.wav" ;select a soundfile
al inch 1      ;Use realtime input

fsig pvsanal  al, ifftsize, ifftsize/4, ifftsize, iwtype
kamp, kfr pvsbin  fsig, 10
adm  oscil  kamp, kfr, 1
```

```
        out      adm
    endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal*, *pvsynth*, *pvsadsyn*

## Credits

Author: Victor Lazzarini  
August 2006

# pvsblur

pvsblur — Average the amp/freq time functions of each analysis channel for a specified time.

## Description

Average the amp/freq time functions of each analysis channel for a specified time (truncated to number of frames). As a side-effect the input pvoc stream will be delayed by that amount.

## Syntax

```
fsig pvsblur fsigin, kblurtime, imaxdel
```

## Performance

*fsig* -- output pv stream

*fsigin* -- input pv stream.

*kblurtime* -- time in secs during which windows will be averaged .

*imaxdel* -- maximum delay time, used for allocating memory used in the averaging operation.

This opcode will blur a pvstream by smoothing the amplitude and frequency time functions (a type of low-pass filtering); the amount of blur will depend on the length of the averaging period, larger blur-times will result in a more pronounced effect.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

Here is an example of the use of the *pvsblur* opcode. It uses the file *pvsblur.csd* [examples/pvsblur.csd].

### Exemple 404. Example of the *pvsblur* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1
```



```
;; example written by joachim heintz 2009

instr 1
  ifftsize =      1024
  ioverlap =      ifftsize / 4
  iwinsize =      ifftsize
  iwinshape =      1; von-Hann window
  Sfile =         "fox.wav"
  ain          soundin Sfile
  fftin        pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of the audio-signal
  fftblur      pvsblur fftin, p4, 1; blur
  aout         pvsynth fftblur; resynthesis
               out      aout
endin

</CsInstruments>
<CsScore>
i 1 0 3 0
i 1 3 3 .1
i 1 6 3 .5
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal, pvsynth, pvsadsyn*

## Credits

Author: Victor Lazzarini  
November 2004

New plugin in version 5

November 2004.

# pvsbuffer

pvsbuffer — This opcode creates and writes to a circular buffer for f-signals (streaming PV signals).

## Description

This opcode sets up and writes to a circular buffer of length *ilen* (secs), giving a handle for the buffer and a time pointer, which holds the current write position (also in seconds). It can be used with one or more *pvsbufread* opcodes. Writing is circular, wrapping around at the end of the buffer.

## Syntax

```
ihandle, ctime pvsbuffer fsig, ilen
```

## Initialisation

### Initialisation

*ihandle* -- handle identifying this particular buffer, which should be passed to a reader opcode.

*ilen* -- buffer length in seconds.

*fsig* -- an input pv stream

*ctime* -- the current time of writing in the buffer

*pvsbuffer* stores *fsig* in a buffer which can be read by *pvsbufread* using the handle *ihandle*. Different buffers will have different handles so they can be read independently by different *pvsbufread* opcodes. *pvsbuffer* gives in its output the current time (*ctime*) inside the ring buffer which has just been written.

## Examples

See *pvsbufread* for examples of the *pvsbuffer* opcode.

## See also

*pvsbufread*

## Credits

Author: Victor Lazzarini  
July 2007

# pvsbufread

pvsbufread — This opcode reads a circular buffer of f-signals (streaming PV signals).

## Description

This opcode sets up and writes to a circular buffer of length *ilen* (secs), giving a handle for the buffer and a time pointer, which holds the current write position (also in seconds). It can be used with one or more pvsbufread opcodes. Writing is circular, wrapping around at the end of the buffer.

## Syntax

```
fsig pvsbufread ktime, khandle[, ilo, ihi]
```

## Initialisation

### Initialisation

*ilo, ihi* -- set the lowest and highest freqs to be read from the buffer (defaults to 0, Nyquist).

*fsig* -- output pv stream

*ktime* -- time position of reading pointer (in secs).

*khandle* -- handle identifying the buffer to be read. When using k-rate handles, it is important to initialise the k-rate variable to a given existing handle. When changing buffers, fsig buffers need to be compatible (same fsig format).

*pvsbufread* reads f-signals from a buffer created by

With this opcode and *pvsbuffer*, it is possible to, among other things:



### Note

It is important that the handle value passed to *pvsbufread* is valid and was created by *pvsbuffer*. Csound will crash with invalid handles.

## Examples

Here is an example of the pvsbufread opcode. It does 'brassage' by switching between two buffers.

### Exemple 405. Example of the pvsbufread opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
fsig1  pvsanal  asig1,1024,256,1024,1
fsig2  pvsanal  asig2,1024,256,1024,1
```

```

ibuf1,kt1  pvsbuffer  fsig1, 10  ; 10-sec buf with fsig1
ibuf2,kt2  pvsbuffer  fsig2, 7   ; 7-sec buf with fsig2

khan init ibuf1      ; initialise handle to buf1

if ktrig > 0 then    ; switch buffers according to trigger
khan = ibuf2
else
khan = ibuf1
endif

fsb pvsbufread kt1, khan ; read buffer

```

Here is an example of the pvsbufread opcode. It uses the file *pvsbufread.csd* [examples/pvsbufread.csd].

### Exemple 406. Example of the pvsbufread opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsbufread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

opcode FileToPvsBuf, iik, Siiii
;;writes an audio file at the first k-cycle to a fft-buffer (via pvsbuffer)
Sfile, ifftsize, ioverlap, iwinsize, iwinshape xin
ktimek      timeinstk
if ktimek == 1 then
ilen      filelen Sfile
kcycles =   ilen * kr; number of k-cycles to write the fft-buffer
kcount    init      0
loop:
ain      soundin Sfile
fftin    pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape
ibuf, ktim pvsbuffer fftin, ilen + (ifftsize / sr)
        loop_lt kcount, 1, kcycles, loop
        xout      ibuf, ilen, ktim
endif
endop

instr 1
ifftsize =      1024
ioverlap =      ifftsize / 4
iwinsize =      ifftsize
iwinshape =     1; von-Hann window
ibuffer, ilen, k0      FileToPvsBuf "fox.wav", ifftsize, ioverlap, iwinsize, iwinshape
ktmpnt      linseg      ilen, p3, 0; reads the buffer backwards in p3 seconds
fread      pvsbufread      ktmpnt, ibuffer
aout      pvsynth fread
        out      aout
endin

```

```
</CsInstruments>
<CsScore>
i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal, pvsynth, pvsbuffer, pvsadsyn*

## Credits

Author: Victor Lazzarini  
July 2007

# pvscale

pvscale — Scale the frequency components of a pv stream.

## Description

Scale the frequency components of a pv stream, resulting in pitch shift. Output amplitudes can be optionally modified in order to attempt formant preservation.

## Syntax

```
fsig pvscale fsignin, kscal[, ikeepform, igain]
```

## Performance

*f`sig`* -- output pv stream

*f`signin`* -- input pv stream

*k`scal`* -- scaling ratio.

*i`keepform`* -- attempt to keep input signal -- -- formants; 0: do not keep formants; 1: keep formants by imposing original amps; 2: keep formants by filtering using the original spec envelope (defaults to 0).

*i`gain`* -- amplitude scaling (defaults to 1).

The quality of the pitch shift will be improved with the use of a Hanning window in the pvoc analysis. Formant preservation is only successful with strong-formant sounds, such as voices and certain instrumental sounds, but also can be used for interesting transformation effects.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 407. Example

```
asig in                                ; get the signal in
fsig pvsanal asig, 1024, 256, 1024, 1 ; analyse it
ftps pvscale fsig, 1.5, 1, 2           ; transpose it keeping formants
atps pvsynth ftps                    ; synthesise it

adp delayr .1                          ; delay original signal
adel deltapn 1024                       ; by 1024 samples
      delayw asig
      out atps+adel                    ; add transposed and original
```

The example above shows a vocal harmoniser. The delay is necessary to time-align the signals, as the analysis-synthesis process will imply a delay of 1024 samples between the analysis input and the synthesis output.

Here is an example of the use of the *pvscale* opcode. It uses the file *pvscale.csd* [examples/pvscale.csd].

### Exemple 408. Example of the *pvscale* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
ifftsize =      1024
ioverlap =      ifftsize / 4
iwinshape =      ifftsize
iwinshape =      1; von-Hann window
Sfile           =      "fox.wav"
ain             soundin Sfile
fftin           pvsanal ain, ifftsize, ioverlap, iwinshape; fft-analysis of the audio-signal
fftblur         pvscale fftin, p4, p5, p6; scale
aout            pvsynth fftblur; resynthesis
               out      aout
endin

</CsInstruments>
<CsScore>
i 1 0 3 1 0 1; original sound
i 1 3 3 1.5 0 2; fifth higher without ...
i 1 6 3 1.5 1 2; ... and with different ...
i 1 9 3 1.5 2 5; ... kinds of formant preservation
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal*, *pvsynth*, *pvsadsyn*

## Credits

Author: Victor Lazzarini  
November 2004

New plugin in version 5

November 2004.

# pvscent

pvscent — Calculate the spectral centroid of a signal.

## Description

Calculate the spectral centroid of a signal from its discrete Fourier transform.

## Syntax

```
kcent pvscent fsig
```

## Performance

*kcent* -- the spectral centroid

*fsig* -- an input pv stream

## Examples

Here is an example of the use of the *pvscent* opcode. It uses the file *pvscent.csd* [examples/pvscent.csd].

### Exemple 409. Example of the *vtinit* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

giSine          ftgen          0, 0, 4096, 10, 1

instr 1
irefrtm =          p4; time for generating new values for the spectral centroid
ifftsize =          1024
ioverlap =          ifftsize / 4
iwinsize =          ifftsize
iwinshape =          1; von-Hann window
;Sfile            =          "flute-C-octave0.wav"
Sfile            =          "fox.wav"
ain              soundin Sfile
fftin            pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of the audio-signal
ktrig            metro          1 / irefrtm
if ktrig == 1 then
kcenter pvscent fftin; spectral center
endif
aout            oscil          .2, kcenter, giSine
                out          aout
endin
```



```
</CsInstruments>
<CsScore>
i 1 0 2.757 .3
i 1 3 2.757 .05
i 1 6 2.757 .005
i 1 9 2.757 .001
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal, pvsynth, pvsadsyn, pvspitch*

## Credits

Author: John ffitch;  
March 2005

New plugin in version 5

March 2005.

# pvcross

pvcross — Performs cross-synthesis between two source fsigs.

## Description

Performs cross-synthesis between two source fsigs.

## Syntax

```
fsig pvcross fsrc, fdest, kamp1, kamp2
```

## Performance

The operation of this opcode is identical to that of *pvcross* (q.v.), except in using *fsigs* rather than analysis files, and the absence of spectral envelope preservation. The amplitudes from *fsrc* and *fdest* (using scale factors *kamp1* for *fsrc* and *kamp2* for *fdest*) are applied to the frequencies of *fsrc*. *kamp1* and *kamp2* must not exceed the range 0 to 1.

With this opcode, cross-synthesis can be performed on real-time audio input, by using *pvsanal* to generate *fsrc* and *fdest*. These must have the same format.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

Here is an example of the use of the *pvcross* opcode. It uses the file *pvcross.csd* [examples/pvcross.csd].

### Exemple 410. Example of the *pvcross* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
odbfs = 1

;; example written by joachim heintz 2009

instr 1
  ipermut =          p4; 1 = change order of soundfiles
```

```

ifftsize =      1024
ioverlap =      ifftsize / 4
iwinsize =      ifftsize
iwinshape =     1; von-Hann window
Sfile1        =      "fox.wav"
Sfile2        =      "wave.wav"
ain1          soundin Sfile1
ain2          soundin Sfile2
fftin1        pvsanal ain1, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file 1
fftin2        pvsanal ain2, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file 2
ktrans        linseg      0, p3, 1; linear transition
if ipermut == 1 then
  fcross       pvcross fftin2, fftin1, ktrans, 1-ktrans
else
  fcross       pvcross fftin1, fftin2, ktrans, 1-ktrans
endif
aout          pvsynth fcross
              out      aout
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 0; frequencies from fox.wav; amplitudes moving from wave to fox
i 1 3 2.757 1; frequencies from wave.wav, amplitudes moving from fox to wave
e
</CsScore>
</CsoundSynthesizer>

```

## See Also

*pvsanal*, *pvsynth*, *pvsadsyn*

## Credits

Author: Richard Dobson  
August 2001

November 2003. Thanks to Kanata Motohashi, fixed the link to the *pvcross* opcode.

New in version 4.13

# pvsdemix

pvsdemix — Spectral azimuth-based de-mixing of stereo sources.

## Description

Spectral azimuth-based de-mixing of stereo sources, with a reverse-panning result. This opcode implements the Azimuth Discrimination and Resynthesis (ADResS) algorithm, developed by Dan Barry (Barry et Al. "Sound Source Separation Azimuth Discrimination and Resynthesis". DAFx'04, Univ. of Napoli). The source separation, or de-mixing, is controlled by two parameters: an azimuth position (*kpos*) and a subspace width (*kwidth*). The first one is used to locate the spectral peaks of individual sources on a stereo mix, whereas the second widens the 'search space', including/excluding the peaks around *kpos*. These two parameters can be used interactively to extract source sounds from a stereo mix. The algorithm is particularly successful with studio recordings where individual instruments occupy individual panning positions; it is, in fact, a reverse-panning algorithm.



### Avertissement

It is unsafe to use the same *f*-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Syntax

```
fsig pvsdemix fleft, fright, kpos, kwidth, ipoints
```

## Performance

*fsig* -- output pv stream

*fleft* -- left channel input pv stream.

*fright* -- right channel pv stream.

*kpos* -- the azimuth target centre position, which will be de-mixed, from left to right ( $-1 \leq kpos \leq 1$ ). This is the reverse pan-pot control.

*kwidth* -- the azimuth subspace width, which will determine the number of points around *kpos* which will be used in the de-mixing process. ( $1 \leq kwidth \leq ipoints$ )

*ipoints* -- total number of discrete points, which will divide each pan side of the stereo image. This ultimately affects the resolution of the process.

## Examples

The example below takes a stereo input and passes through a de-mixing process revealing a source located at *ipos* +/- *iwidth* points. These parameters can be controlled in realtime (e.g. using FLTK widgets or MIDI) for an interactive search of sound sources.

### Exemple 411. Example

```
ifftsize = 1024
iwtype = 1      /* cleaner with hanning window */
ipos = -0.8     /* to the left of the stereo image */
iwidth = 20     /* use peaks of 20 points around it */

al,ar  soundin "sinput.wav"

flc  pvsanal    al, ifftsize, ifftsize/4, ifftsize, iwtype
frc  pvsanal    ar, ifftsize, ifftsize/4, ifftsize, iwtype
fdm  pvsdemix   flc, frc, kpos, kwidth, 100
adm  pvsynth    fdm

      outs      adm,adm
```

## Credits

Author: Victor Lazzarini  
January 2005

New plugin in version 5  
January 2005.

# pvdiskin

pvdiskin — Read a selected channel from a PVOC-EX analysis file.

## Description

Create an fsigr stream by reading a selected channel from a PVOC-EX analysis file, with frame interpolation.

## Syntax

```
fsigr pvdiskin Sfname, ktscal, kgain[, ioffset, ichan]
```

## Initialization

*Sfname* -- Name of the analysis file. This must have the .pvx file extension.

A multi-channel PVOC-EX file can be generated using the extended *pvanal* utility.

*ichan* -- (optional) The channel to read (counting from 1). Default is 1.

*ioff* -- start offset from beginning of file (secs) (default: 0) .

## Performance

*ktscal* -- time scale, ie. the read pointer speed (1 is normal speed, negative is backwards,  $0 < ktscal < 1$  is slower and  $ktscal > 1$  is faster)

*kgain* -- gain scaling.

## Examples

```
fsigr pvdiskin "test.pvx", 1, 1 ; read PVOC-EX file with tscale and gain = 1  
aout pvsynth fsigr ; resynthesize it
```

## Credits

Author: Victor Lazzarini  
May 2007  
New in Csound 5.06

# pvsdisp

pvsdisp — Displays a PVS signal as an amplitude vs. freq graph.

## Description

This opcode will display a PVS signal fsig. Uses X11 or FLTK windows if enabled, else (or if -g flag is set) displays are approximated in ASCII characters.

## Syntax

```
pvsdisp fsig[, ibins, iwtflg]
```

## Initialization

*iprd* -- the period of pvsdisp in seconds.

*ibins* (optional, default=all bins) -- optionally, display only ibins bins.

*iwtflg* (optional, default=0) -- wait flag. If non-zero, each pvsdisp is held until released by the user. The default value is 0 (no wait).

## Performance

*pvsdisp* -- displays the PVS signal frame-by-frame.

## Examples

Here is an example of the pvsdisp opcode. It uses the file *pvsdisp.csd* [examples/pvsdisp.csd]. This example uses realtime input, but you can also use it for soundfile input.

### Exemple 412. Example of the pvsdisp opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsdisp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
asig inch 1
;al soundin "input.wav" ;select a soundifle
fsig pvsanal asig, 1024,256, 1024, 1
```

```
pvsdisp fsig
endin
</CsInstruments>
<CsScore>

i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal, pvsynth, dispfft, print, pvsadsyn*

## Credits

Author: Victor Lazzarini, 2006



# pvsfilter

pvsfilter — Multiply amplitudes of a pvoc stream by those of a second pvoc stream, with dynamic scaling.

## Description

Multiply amplitudes of a pvoc stream by those of a second pvoc stream, with dynamic scaling.

## Syntax

```
fsig pvsfilter fsigin, fsigfil, kdepth[, igain]
```

## Performance

*fsig* -- output pv stream

*fsigin* -- input pv stream.

*fsigfil* -- filtering pvoc stream.

*kdepth* -- controls the depth of filtering of fsigin by fsigfil .

*igain* -- amplitude scaling (optional, defaults to 1).

Here the input pvoc stream amplitudes are modified by the filtering stream, keeping its frequencies intact. As usual, both signals have to be in the same format.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 413. Example

```
kfreq expon 500, p3, 4000 ; 3-octave sweep
kdepth linseg 1, p3/2, 0.5, p3/2, 1 ; varying filter depth

asig in ; input
afil oscili 1, kfreq, 1 ; filter t-domain signal

fim pvsanal asig,1024,256,1024,0 ; pvoc analysis
fil pvsanal afil,1024,256,1024,0
fou pvsfilter fim, fil, kdepth ; filter signal
aout pvsynth fou ; pvoc synthesis
```

In the example above the filter curve will depend on the spectral envelope of *afil*; in the simple case of a sinusoid, it will be equivalent to a narrowband band-pass filter.

Here is an example of the use of the *pvsfilter* opcode. It uses the file *pvsfilter.csd* [examples/pvsfilter.csd].

### Exemple 414. Example of the *pvsfilter* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

giSine          ftgen          0, 0, 4096, 10, 1
giBell          ftgen          0, 0, 4096, 9, .56, 1, 0, .57, .67, 0, .92, 1.8, 0, .93, 1.8, 0, 1.19, 2.0

instr 1
ipermut =          p4; 1 = change order of soundfiles
ifftsize =          1024
ioverlap =          ifftsize / 4
iwinsize =          ifftsize
iwinshape =          1; von-Hann window
Sfile1          =          "fox.wav"
ain1            soundin Sfile1
kfreq           randomi 200, 300, 3
ain2            oscili          .2, kfreq, giBell
;ain2           oscili          .2, kfreq, giSine; try also this
fftin1          pvsanal ain1, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file 1
fftin2          pvsanal ain2, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file 2
if ipermut == 1 then
fcross          pvsfilter fftin2, fftin1, 1
else
fcross          pvsfilter fftin1, fftin2, 1
endif
aout            pvsynth fcross
out             aout * 20
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 0; frequencies from fox.wav, amplitudes multiplied by amplitudes of giBell
i 1 3 2.757 1; frequencies from giBell, amplitudes multiplied by amplitudes of fox.wav
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal*, *pvsynth*, *pvsadsyn*

## Credits

Author: Victor Lazzarini

November 2004

New plugin in version 5

November 2004.

# pvsfread

pvsfread — Read a selected channel from a PVOC-EX analysis file.

## Description

Create an fsig stream by reading a selected channel from a PVOC-EX analysis file loaded into memory, with frame interpolation. Only format 0 files (amplitude+frequency) are currently supported. The operation of this opcode mirrors that of pvoc, but outputs an fsig instead of a resynthesized signal.

## Syntax

```
fsig pvsfread ktimpt, ifn [, ichan]
```

## Initialization

*ifn* -- Name of the analysis file. This must have the .pvx file extension.

A multi-channel PVOC-EX file can be generated using the extended *pvanal* utility.

*ichan* -- (optional) The channel to read (counting from 0). Default is 0.

## Performance

*ktimpt* -- Time pointer into analysis file, in seconds. See the description of the same parameter of *pvoc* for usage.

Note that analysis files can be very large, especially if multi-channel. Reading such files into memory will very likely incur breaks in the audio during real-time performance. As the file is read only once, and is then available to all other interested opcodes, it can be expedient to arrange for a dedicated instrument to preload all such analysis files at startup.

## Examples

```
idur filelen "test.pvx" ; find dur of (stereo) analysis file
kpos line 0,p3,idur ; to ensure we process whole file
fsigr pvsfread kpos,"test.pvx",1 ; create fsig from R channel
```

(NB: as this example shows, the filelen opcode has been extended to accept both old and new analysis file formats).

## Credits

Author: Richard Dobson  
August 2001

New in version 4.13

# pvsfreeze

**pvsfreeze** — Freeze the amplitude and frequency time functions of a pv stream according to a control-rate trigger.

## Description

This opcode 'freezes' the evolution of pvs stream by locking into steady amplitude and/or frequency values for each bin. The freezing is controlled, independently for amplitudes and frequencies, by a control-rate trigger, which switches the freezing 'on' if equal to or above 1 and 'off' if below 1.

## Syntax

```
fsig pvsfreeze fsigin, kfreeza, kfreezf
```

## Performance

*fsig* -- output pv stream

*fsigin* -- input pv stream.

*kfreeza* -- freezing switch for amplitudes. Freezing is on if above or equal to 1 and off if below 1.

*kfcf* -- freezing switch for frequencies. Freezing is on if above or equal to 1 and off if below 1.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 415. Example

```
asig in                                ; input
ktrig oscil 1.5, 0.25, 1                ; trigger
fim pvsanal asigl,1024,256,1024,0      ; pvoc analysis
fou pvsfreeze fim, abs(ktrig), abs(ktrig) ; regular 'freeze' of spectra
aout pvsynth fou                        ; pvoc synthesis
```

In the example above the input signal will be regularly 'frozen' for a short while, as the trigger rises above 1 about every two seconds.

Here is an example of the use of the *pvsfreeze* opcode. It uses the file *pvsfreeze.csd* [examples/pvs-freeze.csd].

### Exemple 416. Example of the *pvsfreeze* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

                seed                0

instr 1
  ifftsize =      1024
  ioverlap =      ifftsize / 4
  iwinsize =      ifftsize
  iwinshape =     1; von-Hann window
  Sfile1        =      "fox.wav"
  ain           soundin Sfile1
  kfreq         randomh .7, 1.1, 3; probability of freezing freqs: 1/4
  kamp          randomh .7, 1.1, 3; idem for amplitudes
  fftin         pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file
  freeze        pvsfreeze fftin, kamp, kfreq; freeze amps or freqs independently
  aout          pvsynth freeze; resynthesize
                out              aout
endin

</CsInstruments>
<CsScore>
r 10
i 1 0 2.757
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal*, *pvsynth*, *pvsadsyn*

## Credits

Author: Victor Lazzarini  
May 2006

New plugin in version 5

May 2006.

# pvsftr

pvsftr — Reads amplitude and/or frequency data from function tables.

## Description

Reads amplitude and/or frequency data from function tables.

## Syntax

```
pvsftr fsrc, ifna [, ifnf]
```

## Initialization

*ifna* -- A table, at least *inbins* in size, that stores amplitude data. Ignored if *ifna* = 0

*ifnf* (optional) -- A table, at least *inbins* in size, that stores frequency data. Ignored if *ifnf* = 0

## Performance

*fsrc* -- a PVOC-EX formatted source.

Enables the contents of *fsrc* to be exchanged with function tables for custom processing. Except when the frame overlap equals *ksmps* (which will generally not be the case), the frame data is not updated each control period. The data in *ifna*, *ifnf* should only be processed when *kflag* is set to 1. To process only frequency data, set *ifna* to zero.

As the function tables are required only to store data from *fsrc*, there is no advantage in defining them in the score, and they should generally be created in the instrument, using *ftgen*.

By exporting amplitude data, say, from one *fsig* and importing it into another, basic cross-synthesis (as in *pvcross*) can be performed, with the option to modify the data beforehand using the table manipulation opcodes.

Note that the format data in the source *fsig* is not written to the tables. This therefore offers a means of transferring amplitude and frequency data between non-identical *fsigs*. Used this way, these opcodes become potentially pathological, and can be relied upon to produce unexpected results. In such cases, re-synthesis using *pvsadsyn* would almost certainly be required.

To perform a straight copy from one *fsig* to another one of identical format, the conventional assignment syntax can be used:

```
fsig1 = fsig2
```

It is not necessary to use function tables in this case.

## Examples

```
ifn      ftgen      0,0,inbins,10,1      ; make ftable
kflag    pvsftw      fsrc,ifn      ; export amps to table,
kamp     init       0
if      kflag==0    kgoto contin      ; only proc when frame is ready
; kill lowest bins, for obvious effect
      tablew      kamp,1,ifn
      tablew      kamp,2,ifn
      tablew      kamp,3,ifn
      tablew      kamp,4,ifn
; read modified data back to fsrc
      pvsftr      fsrc,ifn
contin:
; and resynth
aout     pvsynth     fsrc
```

## See Also

*pvsftw*

## Credits

Author: Richard Dobson  
August 2001

New in version 4.13



# pvsftw

pvsftw — Writes amplitude and/or frequency data to function tables.

## Description

Writes amplitude and/or frequency data to function tables.

## Syntax

```
kflag pvsftw fsrc, ifna [, ifnf]
```

## Initialization

*ifna* -- A table, at least inbins in size, that stores amplitude data. Ignored if *ifna* = 0

*ifnf* -- A table, at least inbins in size, that stores frequency data. Ignored if *ifnf* = 0

## Performance

*kflag* -- A flag that has the value of 1 when new data is available, 0 otherwise.

*fsrc* -- a PVOC-EX formatted source.

Enables the contents of *fsrc* to be exchanged with function tables, for custom processing. Except when the frame overlap equals *ksmps* (which will generally not be the case), the frame data is not updated each control period. The data in *ifna*, *ifnf* should only be processed when *kflag* is set to 1. To process only frequency data, set *ifna* to zero.

As the functions tables are required only to store data from *fsrc*, there is no advantage in defining them in the score. They should generally be created in the instrument using *ftgen*.

By exporting amplitude data, say, from one fsig and importing it into another, basic cross-synthesis (as in *pvcross*) can be performed, with the option to modify the data beforehand using the table manipulation opcodes.

Note that the format data in the source fsig is not written to the tables. This therefore offers a means of transferring amplitude and frequency data between non-identical fsigs. Used this way, these opcodes become potentially pathological, and can be relied upon to produce unexpected results. In such cases, re-synthesis using *pvsadsyn* would almost certainly be required.

To perform a straight copy from one fsig to another one of identical format, the conventional assignment syntax can be used:

```
fsig1 = fsig2
```

It is not necessary to use function tables in this case.

## Examples

```
ifn      ftgen      0,0,inbins,10,1      ; make ftable
kflag    pvsftw     fsrc,ifn             ; export amps to table,
kamp     init       0
if       kflag==0   kgoto contin         ; only proc when frame is ready
; kill lowest bins, for obvious effect
          tablew     kamp,1,ifn
          tablew     kamp,2,ifn
          tablew     kamp,3,ifn
          tablew     kamp,4,ifn
; read modified data back to fsrc
          pvsftr      fsrc,ifn
contin:
; and resynth
aout     pvsynth     fsrc
```

## See Also

*pvsftr*

## Credits

Author: Richard Dobson  
August 2001

New in version 4.13

# pvsfwrite

pvsfwrite — Ecrit un signal fsig dans un fichier PVOCEX.

## Description

Cet opcode écrit un signal fsig dans un fichier PVOCEX (qui peut être lu à son tour par *pvsfread* ou par d'autres programmes qui supportent les fichiers PVOCEX en entrée).

## Syntaxe

```
pvsfwrite fsig, ifile
```

## Initialisation

*fsig* -- données du fsig en entrée. *ifile* -- nom du fichier (une chaîne de caractères entre guillemets) .

## Exemples

Voici un exemple de l'opcode pvsfwrite. Il utilise le fichier *pvsfwrite.csd* [examples/pvsfwrite.csd]. This example uses realtime audio input.

### Exemple 417. Exemple de l'opcode pvsfwrite

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvsfwrite.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

;By Victor Lazzarini 2008

instr 1
asig oscili 10000, 440, 1
fss pvsanal asig, 1024,256,1024,0
pvsfwrite fss, "mypvs.pvx"
ase pvsynth fss
      out ase
endin

instr 2 ; must be called after instr 1 finishes
ktim timeinstd
fss pvsfread ktim, "mypvs.pvx"
asig pvsynth fss
      out asig
endin
```

```
</CsInstruments>
<CsScore>
f1 0 16384 10 1
i1 0 1
i2 1 1
e
</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau plugin dans la version 5

Novembre 2004.

# pvshift

pvshift — Shift the frequency components of a pv stream, stretching/compressing its spectrum.

## Description

Shift the frequency components of a pv stream, stretching/compressing its spectrum.

## Syntax

```
fsig pvshift fsigin, kshift, klowest[, ikeepform, igain]
```

## Performance

*fsig* -- output pv stream

*fsigin* -- input pv stream

*kshift* -- shift amount (in Hz, positive or negative).

*klowest* -- lowest frequency to be shifted.

*ikeepform* -- attempt to keep input signal formants; 0: do not keep formants; 1: keep formants by imposing original amps; 2: keep formants by filtering using the original spec envelope (defaults to 0).

*igain* -- amplitude scaling (defaults to 1).

This opcode will shift the components of a pv stream, from a certain frequency upwards, up or down a fixed amount (in Hz). It can be used to transform a harmonic spectrum into an inharmonic one. The *ikeepform* flag can be used to try and preserve formants for possibly interesting and unusual spectral modifications.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 418. Example

```
asig  in                                ; get the signal in
fsig  pvsanal    asig, 1024, 256, 1024, 1 ; analyse it
ftps  pvshift    fsig, 100, 0             ; add 100 Hz to each component
atps  pvsynth    ftps                    ; synthesise it
```

Depending on the input, this will transform a pitched sound into an inharmonic, bell-like sound.

Here is an example of the use of the *pvshift* opcode. It uses the file *pvshift.csd* [examples/pvshift.csd].

### Exemple 419. Example of the *pvshift* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
ishift      =      p4; shift amount in Hz
ilowest     =      p5; lowest frequency to be shifted
ikeepform =      p6; 0=no formant keeping, 1=keep by amps, 2=keep by spectral envelope
ifftsize    =      1024
ioverlap    =      ifftsize / 4
iwinshape   =      1; von-Hann window
Sfile       =      "fox.wav"
ain         soundin Sfile
fftin       pvsanal ain, ifftsize, ioverlap, iwinshape; fft-analysis of file
fshift      pvshift      fftin, ishift, iwinshape, ikeepform; shift frequencies
aout        pvsynth fshift; resynthesize
out         aout
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 0 0 0; no shift at all
i 1 3 2.757 100 0 0; shift all frequencies by 100 Hz
i 1 6 2.757 200 0 0; by 200 Hz
i 1 9 2.757 200 0 1; keep formants by method 1
i 1 12 2.757 200 0 2; by method 2
i 1 15 2.757 200 1000 0; shift by 200 Hz but just above 1000 Hz
i 1 18 2.757 1000 500 0; shift by 1000 Hz above 500 Hz
i 1 21 2.757 1000 300 0; above 300 Hz
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal*, *pvsynth*, *pvsadsyn*

## Credits

Author: Victor Lazzarini  
November 2004

New plugin in version 5

November 2004.

# pvsifd

pvsifd — Instantaneous Frequency Distribution, magnitude and phase analysis.

## Description

The pvsifd opcode takes an input a-rate signal and performs an Instantaneous Frequency, magnitude and phase analysis, using the STFT and pvsifd (Instantaneous Frequency Distribution), as described in Lazarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc.of ICMC05, Barcelona. It generates two PV streaming signals, one containing the amplitudes and frequencies (a similar output to pvsanal) and another containing amplitudes and unwrapped phases.

## Syntax

```
ffr,fphs pvsifd ain, ifftsize, ihopsize, iwintype[,iscal]
```

## Performance

*f<sub>fr</sub>* -- output pv stream in AMP\_FREQ format

*f<sub>p<sub>h</sub>s</sub>* -- output pv stream in AMP\_PHASE format

*ifftsize* -- FFT analysis size, must be power-of-two and integer multiple of the hopsize.

*ihopsize* -- hopsize in samples

*iwintype* -- window type (0: Hamming, 1: Hanning)

*iscal* -- amplitude scaling (defaults to 1).



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 420. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; pvsifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
aout resyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows the pvsifd analysis feeding into partial tracking and cubic-phase additive re-

synthesis with pitch shifting.

## Credits

Author: Victor Lazzarini  
June 2005

New plugin in version 5

November 2004.



# pvsinfo

pvsinfo — Get information from a PVOC-EX formatted source.

## Description

Get format information about fsrc, whether created by an opcode such as pvsanal, or obtained from a PVOC-EX file by pvsfread. This information is available at init time, and can be used to set parameters for other pvs opcodes, and in particular for creating function tables (e.g. for pvsftw), or setting the number of oscillators for pvsadsyn.

## Syntax

```
ioverlap, inumbins, iwinsize, iformat pvsinfo fsrc
```

## Initialization

*ioverlap* -- The stream overlap size.

*inumbins* -- The number of analysis bins (amplitude+frequency) in fsrc. The underlying FFT size is calculated as (inumbins -1) \* 2.

*iwinsize* -- The analysis window size. May be larger than the FFT size.

*iformat* -- The analysis frame format. If fsrc is created by an opcode, iformat will always be 0, signifying amplitude+frequency. If fsrc is defined from a PVOC-EX file, iformat may also have the value 1 or 2 (amplitude+phase, complex).

## Examples

```
fim      pvsfread  "test.pvx"      ; import pvocex file
iovl,inb,iws,ifmt pvsinfo  fim      ; get inumbins info
ifn      ftgen     0,0,inb,10,1    ; and create f-table
```

## Credits

Author: Richard Dobson  
August 2001

New in version 4.13

# pvsinit

pvsinit — Initialise a spectral (f) variable to zero.

## Description

Fermorms the equavent to an init operation on an f-variable.

## Syntax

```
fsig pvsinit isize[,iolap,iwinsize,iwintype, iformat]
```

## Performance

*fsig* -- output pv stream set to zero.

*isize* -- size of the DFT frame.

*iolap* -- size of the analysis overlap, defaults to isize/4.

*iwinsize* -- size of the analysis window, defaults to isize.

*iwintype* -- type of analysis window, defaults to 1, Hanning.

*iformat* -- pvsdata format, defaults to 0:PVS\_AMP\_FREQ.

## Examples

### Exemple 421. Example

```
fsig pvsinit 1024
```

## Credits

Author: Victor Lazzarini  
November 2004

New plugin in version 5

November 2004.

# pvsin

pvsin — Retrieve an fsig from the input software bus; a pvs equivalent to chani.

## Description

This opcode retrieves an f-sig from the pvs in software bus, which can be used to get data from an external source, using the Csound 5 API. A channel is created if not already existing. The fsig channel is in that case initialised with the given parameters. It is important to note that the pvs input and output (pvsout opcode) busses are independent and data is not shared between them.

## Syntax

```
fsig pvsin kchan[ , isize, iolap, iwinsize, iwintype, iformat]
```

## Initialisation

*isize* -- initial DFT size, defaults to 1024.

*iolap* -- size of overlap, defaults to isize/4.

*isize* -- size of analysis window, defaults to isize.

*isize* -- type of window, defaults to Hanning (1) (see pvsanal)

*isize* -- data format, defaults 0 (PVS\_AMP\_FREQ). Other possible values are 1 (PVS\_AMP\_PHASE), 2 (PVS\_COMPLEX) or 3 (PVS\_TRACKS).

## Performance

*fsig* -- output fsig.

*kchan* -- channel number. If non-existent, a channel will be created.

## Examples

### Exemple 422. Example

```
fsig pvsin 0 ; get data from pvs in bus channel 0
```

## Credits

Author: Victor Lazzarini  
Aoust 2006

# pvsmaska

pvsmaska — Modify amplitudes using a function table, with dynamic scaling.

## Description

Modify amplitudes of fsrc using function table, with dynamic scaling.

## Syntax

```
fsig pvsmaska fsrc, ifn, kdepth
```

## Initialization

*ifn* -- The f-table to use. Given fsrc has N analysis bins, table ifn must be of size N or larger. The table need not be normalized, but values should lie within the range 0 to 1. It can be supplied from the score in the usual way, or from within the orchestra by using *pvsinfo* to find the size of fsrc, (returned by pvsinfo in inbins), which can then be passed to ftgen to create the f-table.

## Performance

*kdepth* -- Controls the degree of modification applied to fsrc, using simple linear scaling. 0 leaves amplitudes unchanged, 1 applies the full profile of ifn.

Note that power-of-two FFT sizes are particularly convenient when using table-based processing, as the number of analysis bins (inbins) is then a power-of-two plus one, for which an exactly matching f-table can be created. In this case it is important that the f-table be created with a size of inbins, rather than as a power of two, as the latter will copy the first table value to the guard point, which is inappropriate for this opcode.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

Here is an example of the use of the *pvsmaska* opcode. It uses the file *pvsmaska.csd* [examples/pvsmaska.csd].

### Exemple 423. Example of the *pvsmaska* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>  
<CsOptions>  
-odac  
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

; function table for defining amplitude peaks (from the example of Richard Dobson)
giTab      ftgen      0, 0, 513, 8, 0, 2, 1, 3, 0, 4, 1, 6, 0, 10, 1, 12, 0, 16, 1, 32, 0, 1, 0,

instr 1
imod      =          p4; degree of midification (0-1)
ifftsize =      1024
ioverlap =      ifftsize / 4
iwinsize =      ifftsize
iwinshape =      1; von-Hann window
Sfile     =      "fox.wav"
ain       soundin Sfile
fftin     pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file
fmask     pvsmaska      fftin, giTab, imod
aout      pvsynth fmask; resynthesize
          out          aout
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 0
i 1 3 2.757 1
e
</CsScore>
</CsoundSynthesizer>

```

## See Also

*pvsanal, pvsynth, pvsadsyn*

## Credits

Author: Richard Dobson  
August 2001

New in version 4.13

# pvsmix

pvsmix — Mix 'seamlessly' two pv signals.

## Description

Mix 'seamlessly' two pv signals. This opcode combines the most prominent components of two pvoc streams into a single mixed stream.

## Syntax

```
fsig pvsmix fsigin1, fsigin2
```

## Performance

*fsig* -- output pv stream

*fsigin1* -- input pv stream.

*fsigin2* -- input pv stream, which must have same format as fsigin1.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 424. Example

```
fsig1 pvsanal asig1,1024,256,1024,0 ; pvoc analysis
fsig2 pvsanal asig2,1024,256,1024,0
fsigout pvsmix fsig1, fsig2 ; mix signals
aout pvsynth fsigout ; pvoc synthesis
```

Depending on the input, this will transform a pitched sound into an inharmonic, bell-like sound.

## Credits

Author: Victor Lazzarini  
November 2004

New plugin in version 5

November 2004.

# pvsmorph

pvsmorph — Performs morphing (or interpolation) between two source fsigs.

## Description

Performs morphing (or interpolation) between two source fsigs.

## Syntax

```
fsig pvsmorph fsig1, fsig2, kampint, kfrqint
```

## Performance

The operation of this opcode is similar to that of *pvinterp* (q.v.), except in using *fsigs* rather than analysis files, and the absence of spectral envelope preservation. The amplitudes and frequencies of *fsig1* are interpolated with those of *fsig2*, depending on the values of *kampint* and *kfrqint*, respectively. These range between 0 and 1, where 0 means *fsig1* and 1, *fsig2*. Anything in between will interpolate amps and/or freqs of the two fsigs.

With this opcode, morphing can be performed on real-time audio input, by using *pvsanal* to generate *fsig1* and *fsig2*. These must have the same format.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

Here is an example of the pvsmorph opcode. It uses the file *pvsmorph.csd* [examples/pvsmorph.csd].

### Exemple 425. Example of the pvsmorph opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac          ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvsmorph.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1
```

```
;; example written by joachim heintz 2009

giSine          ftgen          0, 0, 4096, 10, 1

instr 1
iampint1 =      p4
iampint2 =      p5
ifrqint1 =      p6
ifrqint2 =      p7
kampint linseg   iampint1, p3, iampint2
kfrqint linseg   ifrqint1, p3, ifrqint2
ifftsize =      1024
ioverlap =      ifftsize / 4
iwinsize =      ifftsize
iwinshape =     1; von-Hann window
Sfile1         =      "fox.wav"
ain1           soundin Sfile1
ain2           buzz      .2, 50, 100, giSine
fftin1         pvsanal   ain1, ifftsize, ioverlap, iwinsize, iwinshape
fftin2         pvsanal   ain2, ifftsize, ioverlap, iwinsize, iwinshape
fmorph         pvsmorph  fftin1, fftin2, kampint, kfrqint
aout           pvsynth  fmorph
                out      aout * .5

endin

</CsInstruments>
<CsScore>
i 1 0 3 0 0 1 1
i 1 3 3 1 0 1 0
i 1 6 3 0 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

Here is another example of the pvsmorph opcode. It uses the file *pvsmorph2.csd* [examples/pvs-morph2.csd].

## Exemple 426. Example of the pvsmorph opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009
;; this example uses the files "flute-C-octave0.wav" and
;; "saxophone-alto-C-octave0.wav" from www.archive.org/details/OpenPathMusic44V2

giSine          ftgen          0, 0, 4096, 10, 1

instr 1
iampint1 =      p4; value for interpolating the amplitudes at the beginning ...
iampint2 =      p5; ... and at the end
ifrqint1 =      p6; value for interpolating the frequencies at the beginning ...
ifrqint2 =      p7; ... and at the end
kampint linseg   iampint1, p3, iampint2
kfrqint linseg   ifrqint1, p3, ifrqint2
ifftsize =      1024
ioverlap =      ifftsize / 4
iwinsize =      ifftsize
iwinshape =     1; von-Hann window
Sfile1         =      "flute-C-octave0.wav"
Sfile2         =      "saxophone-alto-C-octave0.wav"
ain1           soundin Sfile1
```



```

ain2          soundin Sfile2
fftin1        pvsanal ain1, ifftsize, ioverlap, iwinshape, iwinshape
fftin2        pvsanal ain2, ifftsize, ioverlap, iwinshape, iwinshape
fmorph        pvsmorph fftin1, fftin2, kampint, kfrqint
aout          pvsynth fmorph
              out      aout * .5

endin

instr 2; moving randomly in certain borders between two spectra
iampintmin =   p4; minimum value for amplitudes
iampintmax =   p5; maximum value for amplitudes
ifrqintmin =   p6; minimum value for frequencies
ifrqintmax =   p7; maximum value for frequencies
imovefreq =    p8; frequency for generating new random values
kampint randomi iampintmin, iampintmax, imovefreq
kfrqint randomi ifrqintmin, ifrqintmax, imovefreq
ifftsize =     1024
ioverlap =     ifftsize / 4
iwinshape =    ifftsize
iwinshape =    1; von-Hann window
Sfile1         =      "flute-C-octave0.wav"
Sfile2         =      "saxophone-alto-C-octave0.wav"
ain1          soundin Sfile1
ain2          soundin Sfile2
fftin1        pvsanal ain1, ifftsize, ioverlap, iwinshape, iwinshape
fftin2        pvsanal ain2, ifftsize, ioverlap, iwinshape, iwinshape
fmorph        pvsmorph fftin1, fftin2, kampint, kfrqint
aout          pvsynth fmorph
              out      aout * .5

endin

</CsInstruments>
<CsScore>
i 1 0 3 0 0 1 1; amplitudes from flute, frequencies from saxophone
i 1 3 3 1 1 0 0; amplitudes from saxophone, frequencies from flute
i 1 6 3 0 1 0 1; amplitudes and frequencies moving from flute to saxophone
i 1 9 3 1 0 1 0; amplitudes and frequencies moving from saxophone to flute
i 2 12 3 .2 .8 .2 .8 5; amps and freqs moving randomly between the two spectra
e
</CsScore>
</CsoundSynthesizer>

```

## See Also

*pvsanal, pvsynth, pvsadsyn*

## Credits

Author: Victor Lazzarini  
 April 2007  
 New in Csound 5.06

# pvsMOOTH

pvsMOOTH — Smooth the amplitude and frequency time functions of a pv stream using parallel 1st order lowpass IIR filters with time-varying cutoff frequency.

## Description

Smooth the amplitude and frequency time functions of a pv stream using a 1st order lowpass IIR with time-varying cutoff frequency. This opcode uses the same filter as the 'tone' opcode, but this time acting separately on the amplitude and frequency time functions that make up a pv stream. The cutoff frequency parameter runs at the control-rate, but unlike tone and tonek, it is not specified in Hz, but as fractions of 1/2 frame-rate (actually the pv stream sampling rate), which is easier to understand. This means that the highest cutoff frequency is 1 and the lowest 0; the lower the frequency the smoother the functions and more pronounced the effect will be. This opcode produces effects that are more or less similar to pvsblur, but with two important differences: 1.smoothing of amplitudes and frequencies use separate sets of filters; and 2. there is no increase in computational cost when higher amounts of 'blurring' (smoothing) are desired.

## Syntax

```
fsig pvsMOOTH fsigin, kacf, kfcf
```

## Performance

*fsig* -- output pv stream

*fsigin* -- input pv stream.

*kacf* -- amount of cutoff frequency for amplitude function filtering, between 0 and 1, in fractions of 1/2 frame-rate.

*kfcf* -- amount of cutoff frequency for frequency function filtering, between 0 and 1, in fractions of 1/2 frame-rate.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 427. Example

```
asigl  in                ; input
fim    pvsanal asigl,1024,256,1024,0 ; pvoc analysis
fou    pvsMOOTH fim, 0.01, 0.01      ; smooth with cf at 1% of 1/2 frame-rate (ca 8.6 Hz)
aout   pvsynth fou                 ; pvoc synthesis
```

In the example above the input signal will be smoothed/blurred by `pvsMOOTH` with a cutoff frequency of 1% of 1/2 frame-rate (which is about 172Hz, so the cf is about 8.6Hz) .

## Credits

Author: Victor Lazzarini  
May 2006

New plugin in version 5

May 2006.

# pvsout

pvsout — Write a fsig to the pvs output bus.

## Description

This opcode writes a fsig to a channel of the pvs output bus. Note that the pvs out bus and the pvs in bus are separate and independent. A new channel is created if non-existent.

## Syntax

```
pvsout fsig, kchan
```

## Performance

*fsig* -- fsig input data.

*kchan* -- pvs out bus channel number.

## Examples

### Exemple 428. Example

```
asig in ; input
fsig pvsanal asig, 1024, 256, 1024, 1 ; analysis
pvsout fsig,0 ; write signal to pvs out bus channel 0
```

## Credits

Author: Victor Lazzarini  
August 2006

# pvsosc

pvsosc — PVS-based oscillator simulator.

## Description

Generates periodic signal spectra in AMP-FREQ format, with the option of four wave types:

1. sawtooth-like (harmonic weight  $1/n$ , where  $n$  is partial number)
2. square-like (similar to 1., but only odd partials)
3. pulse (all harmonics with same weight)
4. cosine

Complex waveforms (ie. all types except cosine) contain all harmonics up to the Nyquist. This makes pvsosc an option for generation of band-limited periodic waves. In addition, types can be changed using a k-rate variable.

## Syntax

```
fsig pvsosc kamp, kfreq, ktype, isize [,ioverlap] [, iwinsize] [, iwintype] [, iformat]
```

## Initialisation

*fsig* -- output pv stream set to zero.

*isize* -- size of analysis frame and window.

*ioverlap* -- (Optional) size of overlap, defaults to  $isize/4$ .

*iwinsize* -- (Optional) window size, defaults to *isize*.

*iwintype* -- (Optional) window type, defaults to Hanning. The choices are currently:

- 0 = Hamming window
- 1 = von Hann window

*iformat* -- (Optional) data format, defaults to 0 which produces AMP:FREQ data. That is currently the only option.

## Performance

*kamp* -- signal amplitude. Note that the actual signal amplitude can, depending on wave type and frequency, vary slightly above or below this value. Generally the amplitude will tend to exceed *kamp* on higher frequencies ( $> 1000$  Hz) and be reduced on lower ones. Also due to the overlap-add process, when resynthesing with pvsynth, frequency glides will cause the output amplitude to fluctuate above and below *kamp*.

*freq* -- fundamental frequency in Hz.

*ktype* -- wave type: 1. sawtooth-like, 2.square-like, 3.pulse and any other value for cosine.

## Examples

Here is an example of the *pvsosc* opcode. It uses the file *pvsosc.csd* [examples/pvsosc.csd].

### Exemple 429. Example of the *pvsosc* opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvsosc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
; a band-limited sawtooth-wave oscillator
fsig pvsosc 10000, 440, 1, 1024 ; generate wave spectral signal
asig pvsynth fsig                ; resynthesise it
out asig
endin

instr 2
; a band-limited square-wave oscillator
fsig pvsosc 10000, 440, 2, 1024 ; generate wave spectral signal
asig pvsynth fsig                ; resynthesise it
out asig
endin

instr 3
; a pulse oscillator
fsig pvsosc 10000, 440, 3, 1024 ; generate wave spectral signal
asig pvsynth fsig                ; resynthesise it
out asig
endin

instr 4
; a cosine-wave oscillator
fsig pvsosc 10000, 440, 4, 1024 ; generate wave spectral signal
asig pvsynth fsig                ; resynthesise it
out asig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 2 2 1
i 3 4 1
i 4 6 1

e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal, pvsynth, pvsadsyn*

## Credits

Author: Victor Lazzarini  
August 2006

# pvspitch

pvspitch — Track the pitch and amplitude of a PVS signal.

## Description

Track the pitch and amplitude of a PVS signal as k-rate variables.

## Syntax

```
kfr, kamp pvspitch fsig, kthresh
```

## Performance

*kamp* -- Amplitude of fundamental frequency

*kfr* -- Fundamental frequency

*fsig* -- an input pv stream

*kthresh* -- analysis threshold (between 0 and 1). Higher values will eliminate low-amplitude components from the analysis.

## Performance

The pitch detection algorithm implemented by *pvspitch* is based upon J. F. Schouten's hypothesis of the neural processes of the brain used to determine the pitch of a sound after the frequency analysis of the basilar membrane. Except for some further considerations, *pvspitch* essentially seeks out the highest common factor of an incoming sound's spectral peaks to find the pitch that may be attributed to it.

In general, input sounds that exhibit pitch will also exhibit peaks in their spectrum according to where their harmonics lie. There are some the exceptions, however. Some sounds whose spectral representation is continuous can impart a sensation of pitch. Such sounds are explained by the centroid or center of gravity of the spectrum and are beyond the scope of the method of pitch detection implemented by *pvspitch* (Using opcodes like *pvscent* might be more appropriate in these cases).

*pvspitch* is able (using a previous analysis *fsig* generated by *pvsanal*) to locate the spectral peaks of a signal. The threshold parameter (*kthresh*) is of utmost importance, as adjusting it can introduce weak yet significant harmonics into the calculation of the fundamental. However, bringing *kthresh* too low would allow harmonically unrelated partials into the analysis algorithm and this will compromise the method's accuracy. These initial steps emulate the response of the basilar membrane by identifying physical characteristics of the input sound. The choice of *kthresh* depends on the actual level of the input signal, since its range (from 0 to 1) spans the whole dynamic range of an analysis bin (from -inf to 0dBFS).

It is important to remember that the input to the *pvspitch* opcode is assumed to be characterised by strong partials within its spectrum. If this is not the case, the results outputted by the opcode may not bear any relation to the pitch of the input signal. If a spectral frame with many unrelated partials was analysed, the greatest common factor of these frequency values that allows for adjacent “harmonics” would be chosen. Thus, noisy frames can be characterised by low frequency outputs of *pvspitch*. This fact allows for a primitive type of instrumental transient detection, as the attack portion of some instrumental tones contain inharmonic components. Should the lowest frequency of the analysed melody be known, then all frequencies detected below this threshold are inaccurate readings, due to the presence of unrelated partials.



In order to facilitate efficient testing of the *pvspitch* algorithm, an amplitude value proportional to the one in the observed in the signal frame is also outputted (*kamp*). The results of *pvspitch* can then be employed to drive an oscillator whose pitch can be audibly compared with that of the original signal (In the example below, an oscillator generates a signal which appears a fifth above the detected pitch).

## Examples

Here is an example of the *pvspitch* opcode. It uses the file *pvspitch.csd* [examples/pvspitch.csd]. This example uses realtime audio input but can be used for audiofile input as well.

### Exemple 430. Example of the *pvspitch* opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvspitch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1

giwave ftgen 0, 0, 4096, 10, 1, 0.5, 0.333, 0.25, 0.2, 0.1666

instr 1

ifftsize = 1024
iwtype = 1 /* cleaner with hanning window */

al inch 1 ;Realtime audio input
;al soundin "input.wav" ;Use this line for file input

fsig pvsanal al, ifftsize, ifftsize/4, ifftsize, iwtype
kfr, kamp pvspitch fsig, 0.01

adm oscil kamp, kfr * 1.5, giwave ;Generate note a fifth above detected pitch
out adm

endin

</CsInstruments>
<CsScore>

i 1 0 30

e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*pvsanal*, *pvsynth*, *pvsadsyn*, *pvscent*

## Credits

Author: Alan OCinneide

August 2005, added by Victor Lazzarini, August 2006

Part of the text has been adapted from the Csound Journal winter 2006 issue's article "Introducing PVS-PITCH: A pitch tracking opcode for Csound" by Alan OCinneide. The article is available at:  
[www.csounds.com/journal/2006winter/pvspitch.html](http://www.csounds.com/journal/2006winter/pvspitch.html)

[<http://www.csounds.com/journal/2006winter/pvspitch.html>]

# pvstencil

pvstencil — Transforms a pvoc stream according to a masking function table.

## Description

Transforms a pvoc stream according to a masking function table; if the pvoc stream amplitude falls below the value of the function for a specific pvoc channel, it applies a gain to that channel.

The pvoc stream amplitudes are compared to a masking table, if they fall below the table values, they are scaled by *kgain*. Prior to the operation, table values are scaled by *klevel*, which can be used as masking depth control.

Tables have to be at least  $\text{fftsize}/2$  in size; for most GENS it is important to use an extended-guard point (size power-of-two plus one), however this is not necessary with GEN43.

One of the typical uses of *pvstencil* would be in noise reduction. A noise print can be analysed with *pval* into a PVOCEX file and loaded in a table with GEN43. This then can be used as the masking table for *pvstencil* and the amount of reduction would be controlled by *kgain*. Skipping post-normalisation will keep the original noise print average amplitudes. This would provide a good starting point for a successful noise reduction (so that *klevel* can be generally set to close to 1).

Other possible transformation effects are possible, such as filtering and 'inverse-masking'.

## Syntax

```
fsig pvstencil fsigin, kgain, klevel, iftable
```

## Performance

*fsig* -- output pv stream

*fsigin* -- input pv stream.

*kgain* -- 'stencil' gain.

*klevel* -- masking function level (scales the ftable prior to 'stenciling').

*iftable* -- masking function table.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 431. Example

```
fsig    pvsanal    asig, 1024, 256, 1024, 1
fclean  pvstencil  fsig, 0, 1, 1
aclean  pvsynth    fclean
```

## Credits

Author: Victor Lazzarini  
November 2004

New plugin in version 5

Nivember 2004.

## pvsvoc

pvsvoc — Combine the spectral envelope of one fsig with the excitation (frequencies) of another.

## Description

This opcode provides support for cross-synthesis of amplitudes and frequencies. It takes the amplitudes of one input fsig and combines with frequencies from another. It is a spectral version of the well-known channel vocoder.

## Syntax

fsig **pvsvoc** famp, fexc, kdepth, kgain

## Performance

*fsig* -- output pv stream

*famp* -- input pv stream from which the amplitudes will be extracted

*fexc* -- input pv stream from which the frequencies will be taken

*kdepth* -- depth of effect, affecting how much of the frequencies will be taken from the second fsig: 0, the output is the famp signal, 1 the output is the famp amplitudes and fexc frequencies.

*kgain* -- gain boost/attenuation applied to the output.



### Avertissement

It is unsafe to use the same f-variable for both input and output of pvs opcodes. Using the same one might lead to undefined behavior on some opcodes. Use a different one on the left and right sides of the opcode.

## Examples

### Exemple 432. Example

```
asig in                                ; get the signal in
asyn oscili 16000, 150, 1              ; excitation signal

famp pvsanal asig, 1024, 256, 1024, 1 ; analyse in signal
fexc pvsanal asyn, 1024, 256, 1024, 1 ; analyse excitation signal
ftps pvsvoc famp, fexc, 1, 1           ; cross it
atps pvsynth ftps                      ; synthesise it

out atps
```

The example above shows a typical cross-synthesis operation. The input signal (say a vocal sound) is

used for its amplitude spectrum. An oscillator with an arbitrary complex waveform produces the excitation signal, giving the vocal sound its pitch.

## Credits

Author: Victor Lazzarini  
April 2005

New plugin in version 5  
April 2005.

# pvsynth

pvsynth — Resynthesise using a FFT overlap-add.

## Description

Resynthesise phase vocoder data (f-signal) using a FFT overlap-add.

## Syntax

```
ares pvsynth fsrc, [iinit]
```

## Performance

*ares* -- output audio signal

*fsrc* -- input signal

*iinit* -- not yet implemented.

## Examples

### Exemple 433. Example (using score-supplied f-table, assuming fsig fftsize = 1024)

```
; score f-table using cubic spline to define shaped peaks
f1 0 513 8 0 2 1 3 0 4 1 6 0 10 1 12 0 16 1 32 0 1 0 436 0

asig buzz      20000,199,50,1      ; pulsewave source
fsig pvsanal asig,1024,256,1024,0 ; create fsig
kmod linseg    0,p3/2,1,p3/2,0    ; simple control sig

fsigout pvsmaska fsig,2,kmod      ; apply weird eq to fsig
aout pvsynth fsigout             ; resynthesize,
    dispfft aout,0.1,1024        ; and view the effect
```

Here is an example of the pvsynth opcode. It uses the file *pvsynth.csd* [examples/pvsynth.csd].

### Exemple 434. Example of the pvsynth opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsynth.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
  ifftsize =      1024
  ioverlap =      ifftsize / 4
  iwinsize =      ifftsize
  iwinshape =      1 ; von-Hann window
  Sfile      =      "fox.wav"
  ain        soundin Sfile
  fftin      pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of the audio-signal
  aout       pvsynth fftin; resynthesis
             out      aout
endin

</CsInstruments>
<CsScore>
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>

```

## See Also

*pvsanal, pvsadsyn*

## Credits

Author: Richard Dobson  
August 2001

New in version 4.13

February 2004. Thanks to a note from Francisco Vila, updated the example.



## pyassign Opcodes

**pyassign** — Assign the value of the given Csound variable to a Python variable possibly destroying its previous content.

### Syntax

```
pyassign "variable", kvalue
```

```
pyassigni "variable", ivalue
```

```
pylassign "variable", kvalue
```

```
pylassigni "variable", ivalue
```

```
pyassignt ktrigger, "variable", kvalue
```

```
pylassignt ktrigger, "variable", kvalue
```

### Description

Assign the value of the given Csound variable to a Python variable possibly destroying its previous content. The resulting Python object will be a float.

### Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

# pycall Opcodes

**pycall** — Invoke the specified Python callable at k-time and i-time (i suffix), passing the given arguments. The call is performed in the global environment, and the result (the returning value) is copied into the Csound output variables specified.

## Syntax

	<b>pycall</b> "callable", karg1, ...
kresult	<b>pycall11</b> "callable", karg1, ...
kresult1, kresult2	<b>pycall12</b> "callable", karg1, ...
kr1, kr2, kr3	<b>pycall13</b> "callable", karg1, ...
kr1, kr2, kr3, kr4	<b>pycall14</b> "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5	<b>pycall15</b> "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6	<b>pycall16</b> "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7	<b>pycall17</b> "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8	<b>pycall18</b> "callable", karg1, ...
	<b>pycall1t</b> ktrigger, "callable", karg1, ...
kresult	<b>pycall11t</b> ktrigger, "callable", karg1, ...
kresult1, kresult2	<b>pycall12t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3	<b>pycall13t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4	<b>pycall14t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5	<b>pycall15t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6	<b>pycall16t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7	<b>pycall17t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8	<b>pycall18t</b> ktrigger, "callable", karg1, ...
	<b>pycall1i</b> "callable", karg1, ...
iresult	<b>pycall11i</b> "callable", iarg1, ...
iresult1, iresult2	<b>pycall12i</b> "callable", iarg1, ...
ir1, ir2, ir3	<b>pycall13i</b> "callable", iarg1, ...

ir1, ir2, ir3, ir4	<b>pycall14i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5	<b>pycall15i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6	<b>pycall16i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7	<b>pycall17i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8	<b>pycall18i</b>	"callable", iarg1, ...
<b>pycalln</b>		"callable", nresults, kresult1, ..., kresultn, karg1, ...
<b>pycallni</b>		"callable", nresults, irestult1, ..., irestultn, iarg1, ...
	<b>pylcall</b>	"callable", karg1, ...
kresult	<b>pylcall1</b>	"callable", karg1, ...
kresult1, kresult2	<b>pylcall12</b>	"callable", karg1, ...
kr1, kr2, kr3	<b>pylcall13</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4	<b>pylcall14</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4, kr5	<b>pylcall15</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6	<b>pylcall16</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7	<b>pylcall17</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8	<b>pylcall18</b>	"callable", karg1, ...
	<b>pylcallt</b>	ktrigger, "callable", karg1, ...
kresult	<b>pylcall1t</b>	ktrigger, "callable", karg1, ...
kresult1, kresult2	<b>pylcall12t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3	<b>pylcall13t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4	<b>pylcall14t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5	<b>pylcall15t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6	<b>pylcall16t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7	<b>pylcall17t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8	<b>pylcall18t</b>	ktrigger, "callable", karg1, ...
	<b>pylcalli</b>	"callable", karg1, ...
irestult	<b>pylcall1i</b>	"callable", iarg1, ...

```

iresult1, irest2                pylcall2i  "callable", iarg1, ...

ir1, ir2, ir3                   pylcall3i  "callable", iarg1, ...

ir1, ir2, ir3, ir4              pylcall4i  "callable", iarg1, ...

ir1, ir2, ir3, ir4, ir5         pylcall5i  "callable", iarg1, ...

ir1, ir2, ir3, ir4, ir5, ir6    pylcall6i  "callable", iarg1, ...

ir1, ir2, ir3, ir4, ir5, ir6, ir7 pylcall7i "callable", iarg1, ...

ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8 pylcall8i "callable", iarg1, ...

pylcalln  "callable", nresults, kresult1, ..., kresultn, karg1, ...

pylcallni "callable", nresults, irest1, ..., irestn, iarg1, ...

```

## Description

This family of opcodes call the specified Python callable at k-time and i-time (i suffix), passing the given arguments. The call is performed in the global environment and the result (the returning value) is copied into the Csound output variables specified.

They pass any number of parameters which are cast to float inside the Python interpreter.

The *pycall/pycalli*, *pycall1/pycall1i* ... *pycall8/pycall8i* opcodes can accomodate for a number of results ranging from 0 to 8 according to their numerical prefix (0 is omitted).

The *pycalln/pycallni* opcodes can accomodate for any number of results: the callable name is followed by the number of output arguments, then come the list of Csound output variable and the list of parameters to be passed.

The returning value of the callable must be `None` for *pycall* or *pycalli*, a float for *pycall1i* or *pycall1i* and a tuple (with proper size) of floats for the *pycall2/pycall2i* ... *pycall8/pycall8i* and *pycalln/pycallni* opcodes.

## Examples

### Exemple 435. Calling a C or Python function

Supposing we have previously defined or imported a function named `get_number_from_pool` as:

```

from random import random, choice

# a pool of 100 numbers
pool = [i ** 1.3 for i in range(100)]

def get_number_from_pool(n, p):
    # substitute an old number with the new number?
    if random() < p:
        i = choice(range(len(pool)))
        pool[i] = n

    # return a random number from the pool
    return choice(pool)

```

then the following orchestra code

```
k2    pycall1 "get_number_from_pool", k1, p6
```

would set k2 randomly from a pool of numbers changing in time. You can pass new pools elements and control the change rate from the orchestra.

### Exemple 436. Calling a Function Object

A more generic implementation of the previous example makes use of a simple function object:

```
from random import random, choice

class GetNumberFromPool:
    def __init__(self, e, begin=0, end=100, step=1):
        self.pool = [i ** e for i in range(begin, end, step)]

    def __call__(self, n, p):
        # substitute an old number with the new number?
        if random() < p:
            i = choice(range(len(pool)))
            pool[i] = n

        # return a random number from the pool
        return choice(pool)

get_number_from_pool1 = GetNumberFromPool(1.3)
get_number_from_pool2 = GetNumberFromPool(1.5, 50, 250, 2)
```

Then the following orchestra code:

```
k2    pycall1 "get_number_from_pool1", k1, p6
k4    pycall1 "get_number_from_pool2", k3, p7
```

would set k2 and k3 randomly from a pool of numbers changing in time. You can pass new pools elements (here k1 and k3) and control the change rate (here p6 and p7) from the orchestra.

As you can see in the first snippet, you can customize the initialization of the pool as well as create several pools.

## Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

## pyeval Opcodes

**pyeval** — Evaluate a generic Python expression and store the result in a Csound variable at k-time or i-time (i suffix).

### Syntax

```
kresult pyeval "expression"

iresult pyevali "expression"

kresult pyleval "expression"

iresult pylevali "expression"

kresult pyevalt ktrigger, "expression"

kresult pylevalt ktrigger, "expression"
```

### Description

These opcodes evaluate a generic Python expression and store the result in a Csound variable at k-time or i-time (i suffix).

The expression must evaluate in a float or an object that can be cast to a float.

They can be used effectively to transfer data from a Python object into a Csound variable.

### Example of the pyleval Opcode Group

The code:

```
k1          pyeval      "v1"
```

will copy the content of the Python variable v1 into the Csound variable k1 at each k-time.

### Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

# pyexec Opcodes

pyexec — Execute a script from a file at k-time or i-time (i suffix).

## Syntax

```
pyexec "filename"

pyexeci "filename"

pylexec "filename"

pylexeci "filename"

pyexec ktrigger, "filename"

plyexec ktrigger, "filename"
```

## Description

Execute a script from a file at k-time or i-time (i suffix).

This is not the same as calling the script with the `system()` call, since the code is executed by the embedded interpreter.

The code contained in the specified file is executed in the global environment for opcodes `pyexec` and `pyexeci` and in the private environment for the opcodes `pylexec` and `pylexeci`.

These opcodes perform no message passing. However, since the statement has access to the main namespace and the private namespace, it can interact with objects previously created in that environment.

The "local" version of the *pyexec* opcodes are useful when the code ran by different instances of an instrument should not interact.

## Example of the pyexec Opcode Group

### Exemple 437. Orchestra (pyexec.orc)

```
sr=44100
kr=4410
ksmps=10
nchnls=1

;If you're not running CsoundAC you need the following line
;to initialize the python interpreter
;pyinit

pyruni "import random"

pyexeci "pyexec1.py"

instr 1

pyexec          "pyexec2.py"
```

```

        pylexeci      "pyexec3.py"
        pylexec      "pyexec4.py"

    endin

```

### Exemple 438. Score (pyexec.sco)

```

il 0 0.01
il 0 0.01

```

### Exemple 439. The pyexec1.py Script

```

import time, os

print
print "Welcome to Csound!"

try:
    s = ', %s?' % os.getenv('USER')
except:
    s = '?'

print 'What sound do you want to hear today%s' % s
answer = raw_input()

```

### Exemple 440. The pyexec2.py script

```

print 'your answer is "%s"' % answer

```

### Exemple 441. The pyexec3.py script

```

message = 'a private random number: %f' % random.random()

```

### Exemple 442. The pyexec4.py script

```

print message

```

If I run this example on my machine I get something like:

```

Using ../../csound.xmg
Csound Version 4.19 (Mar 23 2002)
Embedded Python interpreter version 2.2

```



```
orchname: pyexec.orc
scorename: pyexec.sco
sorting score ...
... done
orch compiler:
11 lines read
      instr 1
Csound Version 4.19 (Mar 23 2002)
displays suppressed

Welcome to Csound!
What sound do you want to hear today, maurizio?
```

then I answer

a sound

then Csound continues with the normal performance

```
your answer is "a sound"
a private random number: 0.884006
new alloc for instr 1:
your answer is "a sound"
a private random number: 0.884006
your answer is "a sound"
a private random number: 0.889868
your answer is "a sound"
a private random number: 0.884006
your answer is "a sound"
a private random number: 0.889868
your answer is "a sound"
a private random number: 0.884006
your answer is "a sound"
...
```

In the same instrument a message is created in the private namespace and printed, appearing different for each instance.

## Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

# pyinit Opcodes

pyinit — Initialize the Python interpreter.

## Syntax

`pyinit`

## Description

In the command-line version of Csound, you must first invoke the *pyinit* opcode in the orchestra header to initialize the Python interpreter, before using any of the other Python opcodes.

But if you use the Python opcodes in the CsoundAC version of Csound, you need not invoke *pyinit*, because CsoundAC automatically initializes the Python interpreter for you. In addition, CsoundAC automatically creates a Python interface to the Csound API, in the form a global instance of the `CsoundAC.CppSound` class named `csound`. Therefore, Python code written in the Csound orchestra has access to the global `csound` object.

## Credits

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

# pyrun Opcodes

pyrun — Run a Python statement or block of statements.

## Syntax

```
pyrun "statement"

pyruni "statement"

pylrun "statement"

pylruni "statement"

pyrunt ktrigger, "statement"

pylrunt ktrigger, "statement"
```

## Description

Execute the specified Python statement at k-time (*pyrun* and *pylrun*) or i-time (*pyruni* and *pylruni*).

The statement is executed in the global environment for *pyrun* and *pyruni* or the local environment for *pylrun* and *pylruni*.

These opcodes perform no message passing. However, since the statement have access to the main namespace and the private namespace, it can interact with objects previously created in that environment.

The "local" version of the *pyrun* opcodes are useful when the code ran by different instances of an instrument should not interact.

## Example of the pyrun Opcode Group

### Exemple 443. Orchestra

```
sr=44100
kr=4410
ksmps=10
nchnls=1

;If you're not running CsoundAC you need the following line
;to initialize the python interpreter
;pyinit

pyruni "import random"

instr 1
    ; This message is stored in the main namespace
    ; and is the same for every instance
    pyruni "message = 'a global random number: %f' % random.random()"
    pyrun "print message"

    ; This message is stored in the private namespace
    ; and is different for different instances
    pylruni "message = 'a private random number: %f' % random.random()"
    pylrun "print message"
```

endin

### **Exemple 444. Score**

```
i1 0 0.1
```

Running this score you should get intermixed pairs of messages from the two instances of instrument 1.

The first message of each pair is stored into the main namespace and so the second instance overwrites the message of the first instance. The result is that first message will be the same for both instances.

The second message is different for the two instances, being stored in the private namespace.

## **Credits**

Copyright (c) 2002 by Maurizio Umberto Puxeddu. All rights reserved. Portions copyright (c) 2004 and 2005 by Michael Gogins. This document has been updated Sunday 25 July 2004 and 1 February 2005 by Michael Gogins.

# rand

rand — Génère une suite contrôlée de nombres aléatoires.

## Description

La sortie est une suite contrôlée de nombres aléatoires entre *-amp* et *+amp*.

## Syntaxe

```
ares rand xamp [, iseed] [, isel] [, ioffset]
```

```
kres rand xamp [, iseed] [, isel] [, ioffset]
```

## Initialisation

*iseed* (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et 1 produira une sortie initiale de *kamp \* iseed*. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système. Avec une valeur négative, la réinitialisation de la graine sera ignorée. La valeur par défaut est 0,5.

*isel* (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

*ioffset* (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

## Exécution

*kamp*, *xamp* -- intervalle sur lequel les nombres aléatoires sont distribués.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de *-kamp* à *+kamp*. *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de *kamp / (racine de 2)*.

## Exemples

Voici un exemple de l'opcode rand. Il utilise le fichier *rand.csd* [examples/rand.csd].

### Exemple 445. Exemple de l'opcode rand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o rand.wav -W ;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Choose a random frequency between 4,100 and 44,100.
kfreq rand 20000
kcps = kfreq + 24100

a1 oscil 30000, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*randh, randi*

## Crédits

Exemple écrit par Kevin Conder.

Grâce à une note de John ffitch, j'ai changé les noms des paramètres.

# randh

randh — Génère des nombres aléatoires et les maintient pendant une certaine durée.

## Description

Génère des nombres aléatoires et les maintient pendant une certaine durée.

## Syntax

```
ares randh xamp, xcps [, iseed] [, isize] [, ioffset]
```

```
kres randh kamp, kcps [, iseed] [, isize] [, ioffset]
```

## Initialisation

*iseed* (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et +1 produira une sortie initiale de *kamp* \* *iseed*. Avec une valeur négative, la réinitialisation de la graine sera ignorée. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système ; c'est la meilleure option pour générer une séquence aléatoire différente à chaque utilisation.

*isize* (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

*ioffset* (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

## Exécution

*kamp*, *xamp* -- intervalle sur lequel les nombres aléatoires sont distribués.

*kcps*, *xcps* -- fréquence à laquelle de nouveaux nombres aléatoires sont générés.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de *-kamp* à *+kamp*. *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de *kamp* / (*racine de 2*).

Les autres unités produisent un bruit à bande limitée : les paramètres *kcps* et *xcps* permettent de choisir un taux de génération des nouveaux nombres aléatoires inférieur aux fréquences d'échantillonnage ou de contrôle. *randh* maintient chaque nouveau nombre durant le cycle spécifié.

## Exemples

Voici un exemple de l'opcode *randh*. Il utilise le fichier *randh.csd* [examples/randh.csd].

### Exemple 446. Exemple de l'opcode randh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o randh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Choose a random frequency between 200 and 1000.
; Generate new random numbers at 4 Hz.
; kamp = 400
; kcps = 4
; iseed = 0.5
; isize = 0
; ioffset = 600

kcps randh 400, 4, 0.5, 0, 600
printk2 kcps

a1 oscil 30000, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*rand, randi*

## Crédits

Exemple écrit par Kevin Conder.



# randi

rand — Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

## Description

Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

## Syntaxe

```
ares randi xamp, xcps [, iseed] [, isize] [, ioffset]
```

```
kres randi kamp, kcps [, iseed] [, isize] [, ioffset]
```

## Initialisation

*iseed* (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et +1 produira une sortie initiale de  $kamp * iseed$ . Avec une valeur négative, la réinitialisation de la graine sera ignorée. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système ; c'est la meilleure option pour générer une séquence aléatoire différente à chaque utilisation.

*isize* (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

*ioffset* (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

## Exécution

*kamp*, *xamp* -- intervalle sur lequel les nombres aléatoires sont distribués.

*kcps*, *xcps* -- fréquence à laquelle de nouveaux nombres aléatoires sont générés.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de  $-kamp$  à  $+kamp$ . *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de  $kamp / (\text{racine de } 2)$ .

Les autres unités produisent un bruit à bande limitée : les paramètres *kcps* et *xcps* permettent de choisir un taux de génération des nouveaux nombres aléatoires inférieur aux fréquences d'échantillonnage ou de contrôle. *randi* produit une interpolation linéaire entre chaque nouveau nombre et le précédent.

## Exemples

Voici un exemple de l'opcode *randi*. Il utilise le fichier *randi.csd* [examples/rand\_i.csd].

### Exemple 447. Exemple de l'opcode randi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o randi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Choose a random frequency between 4,100 and 44,100.
; Generate new random numbers at 10 Hz.
; kamp = 40000
; kcps = 10
; iseed = 0.5
; isize = 0
; ioffset = 4100

kcps randi 40000, 10, 0.5, 0, 4100

a1 oscil 30000, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*rand*, *randh*

## Crédits

Exemple écrit par Kevin Conder.

# random

random — Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale.

## Description

Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale.

## Syntaxe

```
ares random kmin, kmax
```

```
ires random imin, imax
```

```
kres random kmin, kmax
```

## Initialisation

*imin* -- limite inférieure de l'intervalle

*imax* -- limite supérieure de l'intervalle

## Exécution

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

L'opcode *random* est semblable à *linrand* et à *trirand* mais parfois je [Gabriel Maldonado] le trouve plus pratique car il permet de fixer arbitrairement les valeurs du minimum et du maximum.

## Exemples

Voici un exemple de l'opcode random. Il utilise le fichier *random.csd* [examples/random.csd].

### Exemple 448. Exemple de l'opcode random.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o random.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a random number between 220 and 440.
kmin init 220
kmax init 440
k1 random kmin, kmax

printks "k1 = %f\\n", 0.1, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie inclura des lignes comme celles-ci :

```
k1 = 414.232056
k1 = 419.393402
k1 = 275.376373
```

## Voir Aussi

*linrand, randomh, randomi, trirand*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

# randomh

randomh — Génère des nombres aléatoires dans des limites définies par l'utilisateur et les maintient pendant une certaine durée.

## Description

Génère des nombres aléatoires dans des limites définies par l'utilisateur et les maintient pendant une certaine durée.

## Syntaxe

```
ares randomh kmin, kmax, acps
```

```
kres randomh kmin, kmax, kcps
```

## Exécution

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

*kcps*, *acps* -- taux de génération des points aléatoires

L'opcode *randomh* est semblable à *randh* mais il permet à l'utilisateur de fixer arbitrairement les valeurs du minimum et du maximum.

## Exemples

Voici un exemple de l'opcode randomh. Il utilise le fichier *randomh.csd* [examples/randomh.csd].

### Exemple 449. Exemple de l'opcode randomh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc    ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o randomh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Choose a random frequency between 220 and 440 Hz.
; Generate new random numbers at 10 Hz.
```

```
kmin = 220
kmax = 440
kcps = 10

k1 randomh kmin, kmax, kcps

printks "k1 = %f\\n", 0.1, k1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

La sortie inclura des lignes comme celles-ci :

```
k1 = 220.000000
k1 = 414.232056
k1 = 284.095184
```

## Voir Aussi

*randh, random, randomi*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

# randomi

`randomi` — Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

## Description

Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

## Syntaxe

```
ares randomi kmin, kmax, acps
```

```
kres randomi kmin, kmax, kcps
```

## Exécution

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

*kcps*, *acps* -- taux de génération des points aléatoires

L'opcode *randomi* est semblable à *randi* mais il permet à l'utilisateur de fixer arbitrairement les valeurs du minimum et du maximum.

## Exemples

Voici un exemple de l'opcode *randomi*. Il utilise le fichier *randomi.csd* [examples/randomi.csd].

### Exemple 450. Exemple de l'opcode *randomi*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o randomi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Choose a random frequency between 220 and 440.
; Generate new random numbers at 10 Hz.
kmin init 220
```

```
kmax init 440
kcps init 10

k1 randomi kmin, kmax, kcps

printks "k1 = %f\\n", 0.1, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1 = 220.000000
k1 = 414.226196
k1 = 284.101074
```

## Voir Aussi

*randi, random, randomh*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.



# rbjeq

rbjeq — Parametric equalizer and filter opcode with 7 filter types, based on algorithm by Robert Bristow-Johnson.

## Description

Parametric equalizer and filter opcode with 7 filter types, based on algorithm by Robert Bristow-Johnson.

## Syntax

```
ar rbjeq asig, kfco, klvl, kQ, kS[, imode]
```

## Initialization

*imode* ( optional, defaults to zero) - sum of:

- 1: skip initialization (should be used in tied, or re-initialized notes only)

and exactly one of the following values to select filter type:

- 0: resonant lowpass filter. *kQ* controls the resonance: at the cutoff frequency (*kfco*), the amplitude gain is *kQ* (e.g. 20 dB for *kQ* = 10), and higher *kQ* values result in a narrower resonance peak. If *kQ* is set to  $\sqrt{0.5}$  (about 0.7071), there is no resonance, and the filter has a response that is very similar to that of *butterlp*. If *kQ* is less than  $\sqrt{0.5}$ , there is no resonance, and the filter has a -6 dB / octave response from about *kfco* \* *kQ* to *kfco*. Above *kfco*, there is always a -12 dB / octave cutoff.



### NOTE

The *rbjeq* lowpass filter is basically the same as *ar pareq asig, kfco, 0, kQ, 2* but is faster to calculate.

- 2: resonant highpass filter. The parameters are the same as for the lowpass filter, but the equivalent filter is *butterhp* if *kQ* is 0.7071, and "ar *pareq asig, kfco, 0, kQ, 1*" in other cases.
- 4: bandpass filter. *kQ* controls the bandwidth, which is *kfco* / *kQ*, and must be always less than *sr* / 2. The bandwidth is measured between -3 dB points (i.e. amplitude gain = 0.7071), beyond which there is a +/- 6 dB / octave slope. This filter type is very similar to *ar butterbp asig, kfco, kfco / kQ*.
- 6: band-reject filter, with the same parameters as the bandpass filter, and a response similar to that of *butterbr*.
- 8: peaking EQ. It has an amplitude gain of 1 (0 dB) at 0 Hz and *sr* / 2, and *klvl* at the center frequency (*kfco*). Thus, *klvl* controls the amount of boost (if it is greater than 1), or cut (if it is less than 1). Setting *klvl* to 1 results in a flat response. Similarly to the bandpass and band-reject filters, the bandwidth is determined by *kfco* / *kQ* (which must be less than *sr* / 2 again); however, this time it is between  $\sqrt{\text{klvl}}$  points (or, in other words, half the boost or cut in decibels). NOTE: excessively low or high values of *klvl* should be avoided (especially with 32-bit floats), though the opcode was tested with *klvl* = 0.01 and *klvl* = 100. *klvl* = 0 is always an error, unlike in the case of *pareq*, which does allow a zero level.

- 10: low shelf EQ, controlled by *klvl* and *kS* (*kQ* is ignored by this filter type). There is an amplitude gain of *klvl* at zero frequency, while the level of high frequencies (around  $sr / 2$ ) is not changed. At the corner frequency (*kfco*), the gain is  $\sqrt{\text{klvl}}$  (half the boost or cut in decibels). The *kS* parameter controls the steepness of the slope of the frequency response (see below).
- 12: high shelf EQ. Very similar to the low shelf EQ, but affects the high frequency range.

The default value for *imode* is zero (lowpass filter, initialization not skipped).

## Performance

*ar* -- the output signal.

*asig* -- the input signal



### NOTE

If the input contains silent sections, on Intel CPUs a significant slowdown can occur due to denormals. In such cases, it is recommended to process the input signal with "denorm" opcode before filtering it with *rbjeq* (and actually many other filters).

*kfco* -- cutoff, corner, or center frequency, depending on filter type, in Hz. It must be greater than zero, and less than  $sr / 2$  (the range of about  $sr * 0.0002$  to  $sr * 0.49$  should be safe).

*klvl* -- level (amount of boost or cut), as amplitude gain (e.g. 1: flat response, 4: 12 dB boost, 0.1: 20 dB cut); zero or negative values are not allowed. It is recognized by the peaking and shelving EQ types (8, 10, 12) only, and is ignored by other filters.

*kQ* -- resonance (also *kfco* / bandwidth in many filter types). Not used by the shelving EQs (*imode* = 10 and 12). The exact meaning of this parameter depends on the filter type (see above), but it should be always greater than zero, and usually (*kfco* / *kQ*) less than  $sr / 2$ .

*kS* -- shelf slope parameter for shelving filters. Must be greater than zero; a higher value means a steeper slope, with resonance if  $kS > 1$  (however, a too high *kS* value may make the filter unstable). If *kS* is set to exactly 1, the shelf slope is as steep as possible without a resonance. Note that the effect of *kS* - especially if it is greater than 1 - also depends on *klvl*, and it does not have any well defined unit.

## Examples

Here is an example of the *rbjeq* opcode. It uses the file *rbjeq.csd* [examples/rbjeq.csd].

### Exemple 451. An example of the *rbjeq* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rbjeq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr          = 44100
```

```
ksmps = 10
nchnls = 1

instr 1

a1 vco2 10000, 155.6 ; sawtooth wave
kfco expon 8000, p3, 200 ; filter frequency
a1 rbjeq al, kfco, 1, kfco * 0.005, 1, 0 ; resonant lowpass
out a1

endin

</CsInstruments>
<CsScore>

i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Original algorithm by Robert Bristow-Johnson  
Csound orchestra version by Josep M Comajuncosas, Aug 1999  
Converted to C (with optimizations and bug fixes) by Istvan Varga, Dec 2002

# readclock

readclock — Lit la valeur d'une horloge interne.

## Description

Lit la valeur d'une horloge interne.

## Syntaxe

```
ir readclock inum
```

## Initialisation

*inum* -- le numéro d'une horloge. Il y a 32 horloges numérotées de 0 à 31. Toutes les autres valeurs correspondent à l'horloge numéro 32.

*ir* -- valeur, lors de la phase d'initialisation, de l'horloge spécifiée par *inum*.

## Exécution

Entre deux opcodes *clockon* et *clockoff*, le temps CPU utilisé est accumulé dans l'horloge. La précision dépend de la machine et elle est de l'ordre de la milliseconde sur les systèmes UNIX et Windows. L'opcode *readclock* lit la valeur courante d'une horloge pendant une phase d'initialisation.

## Exemples

Voici un exemple de l'opcode *readclock*. Il utilise le fichier *readclock.csd* [examples/readclock.csd].

### Exemple 452. Exemple de l'opcode *readclock*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o readclock.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Start clock #1.
clockon 1
; Do something that keeps Csound busy.
a1 oscili 10000, 440, 1
```

```

out a1
; Stop clock #1.
clockoff 1
; Print the time accumulated in clock #1.
i1 readclock 1
print i1
endin

</CsInstruments>
<CsScore>

; Initialize the function tables.
; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second starting at 0:00.
i 1 0 1
; Play Instrument #1 for one second starting at 0:01.
i 1 1 1
; Play Instrument #1 for one second starting at 0:02.
i 1 2 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

instr 1: i1 = 0.000
instr 1: i1 = 90.000
instr 1: i1 = 180.000

```

## Voir Aussi

*clockoff, clockon*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.56 de Csound

# readk

readk — Lit périodiquement la valeur d'un signal de contrôle de l'orchestre depuis un fichier externe.

## Description

Lit périodiquement la valeur d'un signal de contrôle de l'orchestre depuis un fichier externe dans un format spécifique.

## Syntaxe

```
kres readk ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "readk.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans SSDIR et finalement dans SFDIR.

*iformat* -- spécifie le format des données d'entrée :

- 1 = entiers signés sur 8 bit (char)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII (plein texte)
- 8 = flottants en ASCII (plein texte)

Noter que les formats A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier d'entrée doit être un fichier de données brutes sans en-tête.

*iprd* -- le taux (période) en secondes, arrondi à la période de contrôle de l'orchestre la plus proche, auquel le signal est lu depuis le fichier. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui lira les nouvelles valeurs au taux de contrôle de l'orchestre. Avec des périodes plus longues, les mêmes valeurs seront répétées pendant plus d'une période de contrôle.

## Exécution

*kres* -- le signal lu depuis *ifilename*.

Cette opcode permet de lire la valeur d'un signal généré au taux de contrôle depuis un fichier externe nommé. Le fichier ne doit pas contenir d'en-tête d'information mais il doit contenir une suite temporelle de valeurs de contrôle échantillonnées régulièrement. Pour les formats de texte ASCII, les valeurs doivent être séparées par au moins un espace. Il peut y avoir n'importe quel nombre d'opcodes *readk*

dans un instrument ou dans un orchestre et il peuvent lire à partir du même ou depuis différents fichiers.

## Exemples

Voici un exemple de l'opcode `readk`. Il utilise le fichier *dumpk.csd* [examples/readk.csd].

### Exemple 453. Exemple de l'opcode `readk`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o readk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

0dbfs = 1
; By Andres Cabrera 2008

instr 1
; Read a number from the file every 0.5 seconds
kfibo readk "fibonacci.txt", 7, 0.5
kpitchclass = 8 + ((kfibo % 12)/100)
printk2 kpitchclass
kcps = cpspch( kpitchclass )
printk2 kcps
a1 oscil 0.5, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*dumpk, dumpk2, dumpk3, dumpk4, readk2, readk3, readk4*

## Crédits

Par : John ffitich et Barry Vercoe

1999 ou avant

# readk2

readk2 — Lit périodiquement les valeurs de deux signaux de contrôle de l'orchestre depuis un fichier externe.

## Description

Lit périodiquement les valeurs de deux signaux de contrôle de l'orchestre depuis un fichier externe.

## Syntaxe

```
kr1, kr2 readk2 ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "readk.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans SSDIR et finalement dans SFDIR.

*iformat* -- spécifie le format des données d'entrée :

- 1 = entiers signés sur 8 bit (char)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII (plein texte)
- 8 = flottants en ASCII (plein texte)

Noter que les formats A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier d'entrée doit être un fichier de données brutes sans en-tête.

*iprd* -- le taux (période) en secondes, arrondi à la période de contrôle de l'orchestre la plus proche, auquel les signaux sont lus depuis le fichier. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui lira les nouvelles valeurs au taux de contrôle de l'orchestre. Avec des périodes plus longues, les mêmes valeurs seront répétées pendant plus d'une période de contrôle.

## Exécution

*kr1*, *kr2* -- les signaux lus depuis *ifilename*.

Cette opcode permet de lire les valeurs de deux signaux générés au taux de contrôle depuis un fichier externe nommé. Le fichier ne doit pas contenir d'en-tête d'information mais il doit contenir une suite temporelle de valeurs de contrôle échantillonnées régulièrement. Pour les formats binaires, les échantillons individuels de chaque signal sont alternés. Pour les formats de texte ASCII, les valeurs doivent être sé-



parées par au moins un espace. Les deux "canaux" d'une trame peuvent se trouver sur la même ligne ou être séparés par un caractère de retour à la ligne. Il peut y avoir n'importe quel nombre d'opcodes *readk2* dans un instrument ou dans un orchestre et il peuvent lire à partir du même ou depuis différents fichiers.

## Exemples

Voir l'exemple de *readk*. La seule différence entre *readk* et *readk2* est que *readk2* peut lire deux valeurs à la fois depuis le fichier.

## Voir Aussi

*dumpk, dumpk2, dumpk3, dumpk4, readk, readk3, readk4*

## Crédits

Par : John ffitch et Barry Vercoe

1999 ou avant

# readk3

readk3 — Lit périodiquement les valeurs de trois signaux de contrôle de l'orchestre depuis un fichier externe.

## Description

Lit périodiquement les valeurs de trois signaux de contrôle de l'orchestre depuis un fichier externe.

## Syntaxe

```
kr1, kr2, kr3 readk3 ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "readk.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans SSDIR et finalement dans SFDIR.

*iformat* -- spécifie le format des données d'entrée :

- 1 = entiers signés sur 8 bit (char)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII (plein texte)
- 8 = flottants en ASCII (plein texte)

Noter que les formats A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier d'entrée doit être un fichier de données brutes sans en-tête.

*iprd* -- le taux (période) en secondes, arrondi à la période de contrôle de l'orchestre la plus proche, auquel les signaux sont lus depuis le fichier. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui lira les nouvelles valeurs au taux de contrôle de l'orchestre. Avec des périodes plus longues, les mêmes valeurs seront répétées pendant plus d'une période de contrôle.

## Exécution

*kr1*, *kr2*, *kr3* -- les signaux lus depuis *ifilename*.

Cette opcode permet de lire les valeurs de trois signaux générés au taux de contrôle depuis un fichier externe nommé. Le fichier ne doit pas contenir d'en-tête d'information mais il doit contenir une suite temporelle de valeurs de contrôle échantillonnées régulièrement. Pour les formats binaires, les échantillons individuels de chaque signal sont alternés. Pour les formats de texte ASCII, les valeurs doivent être sé-

parées par au moins un espace. Les trois "canaux" d'une trame peuvent se trouver sur la même ligne ou être séparés par un caractère de retour à la ligne. Il peut y avoir n'importe quel nombre d'opcodes *readk3* dans un instrument ou dans un orchestre et il peuvent lire à partir du même ou depuis différents fichiers.

## Exemples

Voir l'exemple de *readk*. La seule différence entre *readk* et *readk3* est que *readk3* peut lire trois valeurs à la fois depuis le fichier.

## Voir Aussi

*dumpk, dumpk2, dumpk3, dumpk4, readk, readk2, readk4*

## Crédits

Par : John ffitch et Barry Vercoe

1999 ou avant

# readk4

readk4 — Lit périodiquement les valeurs de quatre signaux de contrôle de l'orchestre depuis un fichier externe.

## Description

Lit périodiquement les valeurs de quatre signaux de contrôle de l'orchestre depuis un fichier externe.

## Syntaxe

```
kr1, kr2, kr3, kr4 readk4 ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "readk.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans SSDIR et finalement dans SFDIR.

*iformat* -- spécifie le format des données d'entrée :

- 1 = entiers signés sur 8 bit (char)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII (plein texte)
- 8 = flottants en ASCII (plein texte)

Noter que les formats A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier d'entrée doit être un fichier de données brutes sans en-tête.

*iprd* -- le taux (période) en secondes, arrondi à la période de contrôle de l'orchestre la plus proche, auquel les signaux sont lus depuis le fichier. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui lira les nouvelles valeurs au taux de contrôle de l'orchestre. Avec des périodes plus longues, les mêmes valeurs seront répétées pendant plus d'une période de contrôle.

## Exécution

*kr1, kr2, kr3, kr4* -- les signaux lus depuis *ifilename*.

Cette opcode permet de lire les valeurs de quatre signaux générés au taux de contrôle depuis un fichier externe nommé. Le fichier ne doit pas contenir d'en-tête d'information mais il doit contenir une suite temporelle de valeurs de contrôle échantillonnées régulièrement. Pour les formats binaires, les échantillons individuels de chaque signal sont alternés. Pour les formats de texte ASCII, les valeurs doivent

être séparées par au moins un espace. Les quatre "canaux" d'une trame peuvent se trouver sur la même ligne ou être séparés par un caractère de retour à la ligne. Il peut y avoir n'importe quel nombre d'opcodes *readk4* dans un instrument ou dans un orchestre et il peuvent lire à partir du même ou depuis différents fichiers.

## Exemples

Voir l'exemple de *readk*. La seule différence entre *readk* et *readk4* est que *readk4* peut lire quatre valeurs à la fois depuis le fichier.

## Voir Aussi

*dumpk, dumpk2, dumpk3, dumpk4, readk, readk2, readk3*

## Crédits

Par : John ffitch et Barry Vercoe

1999 ou avant

# reinit

reinit — Suspend une exécution tandis que se déroule une phase spéciale d'initialisation.

## Description

Suspend une exécution tandis que se déroule une phase spéciale d'initialisation.

Chaque fois que cette instruction est rencontrée durant une phase d'exécution, celle-ci est temporairement suspendue tandis qu'une phase spéciale d'initialisation, commençant à *label* et allant jusqu'à *return* ou *endin*, a lieu. L'exécution reprend ensuite à partir de l'endroit où elle fut interrompue.

## Syntaxe

```
reinit label
```

## Exemples

Les instructions suivantes génèrent un signal de contrôle exponentiel dont les valeurs vont de 440 à 880 exactement dix fois pendant la durée p3. Elles utilisent le fichier *reinit.csd* [examples/reinit.csd].

### Exemple 454. Exemple de l'opcode reinit.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reinit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1

reset:
    timeout 0, p3/10, contin
    reinit reset

contin:
    kLine expon 440, p3/10, 880
    aSig oscil 10000, kLine, 1
    out aSig
    rireturn

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1
```

```
i1 0 10  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*rigoto, rireturn*

# release

release — Indicates whether a note is in its « release » stage.

## Description

Provides a way of knowing when a note off message for the current note is received. Only a noteoff message with the same MIDI note number as the one which triggered the note will be reported by *release*.

## Syntax

kflag **release**

## Performance

*kflag* -- indicates whether the note is in its « release » stage. (1 if a note off is received, otherwise 0)

*release* outputs current note state. If current note is in the « release » stage (i.e. if its duration has been extended with *xtratim* opcode and if it has only just deactivated), then the *kflag* output argument is set to 1. Otherwise (in sustain stage of current note), *kflag* is set to 0.

This opcode is useful for implementing complex release-oriented envelopes. When used in conjunction with *xtratim* it can provide an alternative to the hard-coded behaviour of the "r" opcodes (*linsegr*, *expsegr* et al), where release time is set to the longest time specified in the active instrument.

## Examples

See the examples for *xtratim*.

## See Also

*xtratim*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.47



# remoteport

remoteport — Defines the port for use with the remote system.

## Description

Defines the port for use with the *insremot*, *midremot*, *insglobal* and *midglobal* opcodes.

## Syntax

```
remoteport iportnum
```

## Initialization

*iportnum* -- number of the port to be used. If zero or negative the default port 40002 is selected.

## Credits

Author: John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Novemer, 2006

New in Csound version 5.05

# remove

remove — Supprime la définition d'un instrument.

## Description

Supprime la définition d'un instrument tant qu'il n'est pas utilisé.

## Syntaxe

```
remove insnum
```

## Initialisation

*insnum* -- numéro ou nom de l'instrument à effacer

## Exécution

Tant que l'instrument indiqué n'est pas actif, *remove* efface l'instrument et la mémoire qui lui est associée. A employer avec précaution car son utilisation peut conduire à un plantage dans certains cas.

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Juin, 2006

Nouveau dans la version 5.04 de Csound

# repluck

repluck — Modèle physique de corde pincée.

## Description

*repluck* est une implémentation du modèle physique de corde pincée. On peut contrôler le point d'excitation, le point de lecture, le filtre, et un signal audio additionnel, *axcite*. *axcite* est utilisé pour exciter la "corde". Basé sur l'algorithme de Karplus-Strong.

## Syntaxe

ares **repluck** iplk, kamp, icps, kpick, krefl, excite

## Initialisation

*iplk* -- Le point d'excitation est *iplk*, qui représente une fraction de la longueur de la corde (0 à 1). Un point d'excitation de zéro signifie l'absence d'excitation initiale.

*icps* -- La corde produit une hauteur de *icps*.

## Exécution

*kamp* -- Amplitude de la note.

*kpick* -- Fraction de la longueur de la corde où sera lue la sortie.

*krefl* -- le coefficient de réflexion, indiquant l'amortissement et le taux d'extinction. Il doit être strictement compris entre 0 et 1 (il n'acceptera pas 0 ni 1).

## Exécution

*axcite* -- Un signal d'excitation de la corde.

## Exemples

Voici un exemple de l'opcode *repluck*. Il utilise le fichier *repluck.csd* [examples/repluck.csd].

### Exemple 455. Exemple de l'opcode *repluck*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o repluck.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iplk = 0.75
  kamp = 30000
  icps = 220
  kpick = 0.75
  krefl = 0.5
  axcite oscil 1, 1, 1

  apluck repluck iplk, kamp, icps, kpick, krefl, axcite

  out apluck
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*wgpluck2*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
1997

Nouveau dans la version 3.47

# reson

reson — Un filtre à résonance du second ordre.

## Description

Un filtre à résonance du second ordre.

## Syntaxe

```
ares reson asig, kcf, kbw [, iscl] [, iskip]
```

## Initialisation

*iscl* (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ares* -- le signal de sortie au taux audio.

*asig* -- le signal d'entrée au taux audio.

*kcf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*kbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*reson* est un filtre de second ordre dans lequel *kcf* contrôle la fréquence centrale, ou position fréquentielle de la crête de la réponse, et *kbw* contrôle sa largeur de bande (la différence en fréquence entre les points haut et bas à mi-puissance).

## Exemples

Voici un exemple de l'opcode *reson*. Il utilise le fichier *reson.csd* [exemples/reson.csd].

### Exemple 456. Exemple de l'opcode *reson*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a sine waveform.
asine buzz 15000, 440, 3, 1

; Vary the cut-off frequency from 220 to 1280.
kcf line 220, p3, 1320
kbw init 20

; Run the sine through a resonant filter.
ares reson asine, kcf, kbw

; Give the filtered signal the same amplitude
; as the original signal.
al balance ares, asine
out al
endin

</CsInstruments>
<CsScore>

; Table #1, an ordinary sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 4 seconds.
i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*areson, aresonk, atone, atonek, port, portk, resonk, tone, tonek*

## Crédits

Exemple écrit par Kevin Conder.

# resonk

resonk — Un filtre à résonance du second ordre.

## Description

Un filtre à résonance du second ordre.

## Syntaxe

```
kres resonk ksig, kcf, kbw [, iscl] [, iskip]
```

## Initialisation

*iscl* (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*kcf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*kbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*resonk* est semblable à *reson* à part le fait que sa sortie se fait au taux de contrôle plutôt qu'au taux audio.

## Voir Aussi

*areson*, *aresonk*, *atone*, *atonek*, *port*, *portk*, *reson*, *tone*, *tonek*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# resonr

resonr — A bandpass filter with variable frequency response.

## Description

Implementations of a second-order, two-pole two-zero bandpass filter with variable frequency response.

## Syntax

```
ares resonr asig, kcf, kbw [, iscl] [, iskip]
```

## Initialization

The optional initialization variables for *resonr* are identical to the i-time variables for *reson*.

*iscl* (optional, default=0) -- coded scaling factor for resonators. A value of 1 signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A value of 2 raises the response factor so that its overall RMS value equals 1. This intended equalization of input and output power assumes all frequencies are physically present; hence it is most applicable to white noise. A zero value signifies no scaling of the signal, leaving that to some later adjustment (see *balance*). The default value is 0.

*iskip* (optional, default=0) -- initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*asig* -- input signal to be filtered

*kcf* -- cutoff or resonant frequency of the filter, measured in Hz

*kbw* -- bandwidth of the filter (the Hz difference between the upper and lower half-power points)

*resonr* and *resonz* are variations of the classic two-pole bandpass resonator (*reson*). Both filters have two zeroes in their transfer functions, in addition to the two poles. *resonz* has its zeroes located at  $z = 1$  and  $z = -1$ . *resonr* has its zeroes located at  $+\sqrt{R}$  and  $-\sqrt{R}$ , where  $R$  is the radius of the poles in the complex  $z$ -plane. The addition of zeroes to *resonr* and *resonz* results in the improved selectivity of the magnitude response of these filters at cutoff frequencies close to 0, at the expense of less selectivity of frequencies above the cutoff peak.

*resonr* and *resonz* are very close to constant-gain as the center frequency is swept, resulting in a more efficient control of the magnitude response than with traditional two-pole resonators such as *reson*.

*resonr* and *resonz* produce a sound that is considerably different from *reson*, especially for lower center frequencies; trial and error is the best way of determining which resonator is best suited for a particular application.

## Examples

Here is an example of the *resonr* and *resonz* opcodes. It uses the file *resonr.csd* [examples/resonr.csd].



**Exemple 457. Example of the resonr and resonz opcodes.**

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o resonr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Sean Costello */
; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
; The outputs of reson, resonr, and resonz are scaled by coefficients
; specified in the score, so that each filter can be heard on its own
; from the same instrument.

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1

    idur      =      p3
    ibegfreq  =      p4                      ; beginning of sweep frequency
    iendfreq  =      p5                      ; ending of sweep frequency
    ibw       =      p6                      ; bandwidth of filters in Hz
    ifreq     =      p7                      ; frequency of gbuzz that is to be filtered
    iamp      =      p8                      ; amplitude to scale output by
    ires      =      p9                      ; coefficient to scale amount of reson in output
    iresr     =      p10                     ; coefficient to scale amount of resonr in output
    iresz     =      p11                     ; coefficient to scale amount of resonz in output

; Frequency envelope for reson cutoff
    kfreq     linseg ibegfreq, idur * .5, iendfreq, idur * .5, ibegfreq

; Amplitude envelope to prevent clicking
    kenv      linseg 0, .1, iamp, idur - .2, iamp, .1, 0

; Number of harmonics for gbuzz scaled to avoid aliasing
    iharms    =      (sr*.4)/ifreq

    asig      gbuzz 1, ifreq, iharms, 1, .9, 1      ; "Sawtooth" waveform
    ain       =      kenv * asig                  ; output scaled by amp envelope
    ares      reson ain, kfreq, ibw, 1
    aresr     resonr ain, kfreq, ibw, 1
    aresz     resonz ain, kfreq, ibw, 1

    out ares * ires + aresr * iresr + aresz * iresz

endin

</CsInstruments>
<CsScore>

/* Written by Sean Costello */
f1 0 8192 9 1 1 .25                      ; cosine table for gbuzz generator

i1 0 10 1 3000 200 100 4000 1 0 0          ; reson  output with bw = 200
i1 10 10 1 3000 200 100 4000 0 1 0        ; resonr output with bw = 200
i1 20 10 1 3000 200 100 4000 0 0 1        ; resonz output with bw = 200
i1 30 10 1 3000 50 200 8000 1 0 0         ; reson  output with bw = 50
i1 40 10 1 3000 50 200 8000 0 1 0         ; resonr output with bw = 50
i1 50 10 1 3000 50 200 8000 0 0 1         ; resonz output with bw = 50
e

</CsScore>
</CsoundSynthesizer>
```

## Technical History

*resonr* and *resonz* were originally described in an article by Julius O. Smith and James B. Angell.<sup>1</sup> Smith and Angell recommended the *resonz* form (zeros at +1 and -1) when computational efficiency was the main concern, as it has one less multiply per sample, while *resonr* (zeroes at + and - the square root of the pole radius R) was recommended for situations when a perfectly constant-gain center peak was required.

Ken Steiglitz, in a later article<sup>2</sup>, demonstrated that *resonz* had constant gain at the true peak of the filter, as opposed to *resonr*, which displayed constant gain at the pole angle. Steiglitz also recommended *resonz* for its sharper notches in the gain curve at zero and Nyquist frequency. Steiglitz's recent book<sup>3</sup> features a thorough technical discussion of *reson* and *resonz*, while Dodge and Jerse's textbook<sup>4</sup> illustrates the differences in the response curves of *reson* and *resonz*.

## References

1. Smith, Julius O. and Angell, James B., "A Constant-Gain Resonator Tuned by a Single Coefficient," *Computer Music Journal*, vol. 6, no. 4, pp. 36-39, Winter 1982.
2. Steiglitz, Ken, "A Note on Constant-Gain Digital Resonators," *Computer Music Journal*, vol. 18, no. 4, pp. 8-10, Winter 1994.
3. Ken Steiglitz, *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music*. Addison-Wesley Publishing Company, Menlo Park, CA, 1996.
4. Dodge, Charles and Jerse, Thomas A., *Computer Music: Synthesis, Composition, and Performance*. New York: Schirmer Books, 1997, 2nd edition, pp. 211-214.

## See Also

*resonz*

## Credits

Author: Sean Costello  
Seattle, Washington  
1999

New in Csound version 3.55

# resonx

resonx — Emule une série de filtres utilisant l'opcode *reson*.

## Description

*resonx* est équivalent à un filtre constitué de plusieurs couches de filtres *reson* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupure plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de *k* cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

## Syntaxe

```
ares resonx asig, kcf, kbw [, inumlayer] [, iscl] [, iskip]
```

## Initialisation

*inumlayer* (optional) -- (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

*iscl* (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée

*kcf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*kbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

## Voir Aussi

*atonex*, *tonex*

## Crédits

Auteur : Gabriel Maldonado (adapté par John ffitch)  
Italie

Nouveau dans la version 3.49 de Csound

# resonxk

resonxk — Control signal resonant filter stack.

## Description

*resonxk* is equivalent to a group of *resonk* filters, with the same arguments, serially connected. Using a stack of a larger number of filters allows a sharper cutoff.

## Syntax

```
kres resonxk ksig, kcf, kbw[, inumlayer, iscl, istor]
```

## Initialization

*inumlayer* - number of elements of filter stack. Default value is 4. Maximum value is 10

*iscl* (optional, default=0) - coded scaling factor for resonators. A value of 1 signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A value of 2 raises the response factor so that its overall RMS value equals 1. (This intended equalization of input and output power assumes all frequencies are physically present; hence it is most applicable to white noise.) A zero value signifies no scaling of the signal, leaving that to some later adjustment (see *balance*). The default value is 0.

*istor* (optional, default=0) -- initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*kres* - output signal

*ksig* - input signal

*kcf* - the center frequency of the filter, or frequency position of the peak response.

*kbw* - bandwidth of the filter (the Hz difference between the upper and lower half-power points)

*resonxk* is a lot faster than using individual instances in Csound orchestra of the old opcodes, because only one initialization and 'k' cycle are needed at a time, and the audio loop falls entirely inside the cache memory of processor.

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# resony

resony — A bank of second-order bandpass filters, connected in parallel.

## Description

A bank of second-order bandpass filters, connected in parallel.

## Syntax

```
ares resony asig, kbf, kbw, inum, ksep [, isepmode] [, iscl] [, iskip]
```

## Initialization

*inum* -- number of filters

*isepmode* (optional, default=0) -- if *isepmode* = 0, the separation of center frequencies of each filter is generated logarithmically (using octave as unit of measure). If *isepmode* not equal to 0, the separation of center frequencies of each filter is generated linearly (using Hertz). Default value is 0.

*iscl* (optional, default=0) -- coded scaling factor for resonators. A value of 1 signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A value of 2 raises the response factor so that its overall RMS value equals 1. (This intended equalization of input and output power assumes all frequencies are physically present; hence it is most applicable to white noise.) A zero value signifies no scaling of the signal, leaving that to some later adjustment (e.g. *balance*). The default value is 0.

*iskip* (optional, default=0) -- initial disposition of internal data space. Since filtering incorporates a feed-back loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*asig* -- audio input signal

*kbf* -- base frequency, i.e. center frequency of lowest filter in Hz

*kbw* -- bandwidth in Hz

*ksep* -- separation of the center frequency of filters in octaves

*resony* is a bank of second-order bandpass filters, with k-rate variant frequency separation, base frequency and bandwidth, connected in parallel (i.e. the resulting signal is a mix of the output of each filter). The center frequency of each filter depends of *kbf* and *ksep* variables. The maximum number of filters is set to 100.

## Examples

Here is an example of the *resony* opcode. It uses the file *resony.csd* [examples/resony.csd], and *beats.wav* [examples/beats.wav].

## Exemple 458. Example of the resony opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o resony.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a nice sawtooth waveform.
asig vco 32000, 220, 1

; Vary the base frequency from 60 to 600 Hz.
kbf line 60, p3, 600
kbw = 50
inum = 2
ksep = 1
isepmode = 0
iscl = 1

al resony asig, kbf, kbw, inum, ksep, isepmode, iscl

out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Gabriel Maldonado  
Italy  
1999

Example written by Kevin Conder.

New in Csound version 3.56

# resonz

resonz — A bandpass filter with variable frequency response.

## Description

Implementations of a second-order, two-pole two-zero bandpass filter with variable frequency response.

## Syntax

```
ares resonz asig, kcf, kbw [, iscl] [, iskip]
```

## Initialization

The optional initialization variables for *resonr* and *resonz* are identical to the i-time variables for *reson*.

*iscl* -- initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

*iscl* -- coded scaling factor for resonators. A value of 1 signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A value of 2 raises the response factor so that its overall RMS value equals 1. This intended equalization of input and output power assumes all frequencies are physically present; hence it is most applicable to white noise. A zero value signifies no scaling of the signal, leaving that to some later adjustment (see *balance*). The default value is 0.

## Performance

*resonr* and *resonz* are variations of the classic two-pole bandpass resonator (*reson*). Both filters have two zeroes in their transfer functions, in addition to the two poles. *resonz* has its zeroes located at  $z = 1$  and  $z = -1$ . *resonr* has its zeroes located at  $+\sqrt{R}$  and  $-\sqrt{R}$ , where  $R$  is the radius of the poles in the complex  $z$ -plane. The addition of zeroes to *resonr* and *resonz* results in the improved selectivity of the magnitude response of these filters at cutoff frequencies close to 0, at the expense of less selectivity of frequencies above the cutoff peak.

*resonr* and *resonz* are very close to constant-gain as the center frequency is swept, resulting in a more efficient control of the magnitude response than with traditional two-pole resonators such as *reson*.

*resonr* and *resonz* produce a sound that is considerably different from *reson*, especially for lower center frequencies; trial and error is the best way of determining which resonator is best suited for a particular application.

*asig* -- input signal to be filtered

*kcf* -- cutoff or resonant frequency of the filter, measured in Hz

*kbw* -- bandwidth of the filter (the Hz difference between the upper and lower half-power points)

## Technical History

*resonr* and *resonz* were originally described in an article by Julius O. Smith and James B. Angell.<sup>1</sup>



Smith and Angell recommended the *resonz* form (zeros at +1 and -1) when computational efficiency was the main concern, as it has one less multiply per sample, while *resonr* (zeroes at + and - the square root of the pole radius R) was recommended for situations when a perfectly constant-gain center peak was required.

Ken Steiglitz, in a later article <sup>2</sup>, demonstrated that *resonz* had constant gain at the true peak of the filter, as opposed to *resonr*, which displayed constant gain at the pole angle. Steiglitz also recommended *resonz* for its sharper notches in the gain curve at zero and Nyquist frequency. Steiglitz's recent book <sup>3</sup> features a thorough technical discussion of *reson* and *resonz*, while Dodge and Jerse's textbook <sup>4</sup> illustrates the differences in the response curves of *reson* and *resonz*.

## References

1. Smith, Julius O. and Angell, James B., "A Constant-Gain Resonator Tuned by a Single Coefficient," *Computer Music Journal*, vol. 6, no. 4, pp. 36-39, Winter 1982.
2. Steiglitz, Ken, "A Note on Constant-Gain Digital Resonators," *Computer Music Journal*, vol. 18, no. 4, pp. 8-10, Winter 1994.
3. Ken Steiglitz, *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music*. Addison-Wesley Publishing Company, Menlo Park, CA, 1996.
4. Dodge, Charles and Jerse, Thomas A., *Computer Music: Synthesis, Composition, and Performance*. New York: Schirmer Books, 1997, 2nd edition, pp. 211-214.

## See Also

*resonr*

## Credits

Author: Sean Costello  
Seattle, Washington  
1999

New in Csound version 3.55

# resyn

resyn — Streaming partial track additive synthesis with cubic phase interpolation with pitch control and support for timescale-modified input

## Description

The resyn opcode takes an input containing a TRACKS pv streaming signal (as generated, for instance by partials). It resynthesises the signal using linear amplitude and cubic phase interpolation to drive a bank of interpolating oscillators with amplitude and pitch scaling controls. Resyn is a modified version of sin-syn, allowing for the resynthesis of data with pitch and timescale changes.

## Syntax

```
asig resyn fin, kscal, kpitch, kmaxtracks, ifn
```

## Performance

*asig* -- output audio rate signal

*fin* -- input pv stream in TRACKS format

*kscal* -- amplitude scaling

*kpitch* -- pitch scaling

*kmaxtracks* -- max number of tracks in resynthesis. Limiting this will cause a non-linear filtering effect, by discarding newer and higher-frequency tracks (tracks are ordered by start time and ascending frequency, respectively)

*ifn* -- function table containing one cycle of a sinusoid (sine or cosine)

## Examples

### Exemple 459. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
aout resyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows partial tracking of an ifd-analysis signal and cubic-phase additive resynthesis with pitch shifting.

## Credits

Author: Victor Lazzarini  
June 2005

New plugin in version 5

November 2004.

# reverb

reverb — Reverberates an input signal with a « natural room » frequency response.

## Description

Reverberates an input signal with a « natural room » frequency response.

## Syntax

```
ares reverb asig, krvt [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- initial disposition of delay-loop data space (cf. *reson*). The default value is 0.

## Performance

*krvt* -- the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude).

A standard *reverb* unit is composed of four *comb* filters in parallel followed by two *alpass* units in series. Loop times are set for optimal « natural room response. » Core storage requirements for this unit are proportional only to the sampling rate, each unit requiring approximately 3K words for every 10KC. The *comb*, *alpass*, *delay*, *tone* and other Csound units provide the means for experimenting with alternate reverberator designs.

Since output from the standard *reverb* will begin to appear only after 1/20 second or so of delay, and often with less than three-fourths of the original power, it is normal to output both the source and the reverberated signal. If *krvt* is inadvertently set to a non-positive number, *krvt* will be reset automatically to 0.01. (New in Csound version 4.07.) Also, since the reverberated sound will persist long after the cessation of source events, it is normal to put *reverb* in a separate instrument to which sound is passed via a *global variable*, and to leave that instrument running throughout the performance.

## Examples

Here is an example of the reverb opcode. It uses the file *reverb.csd* [examples/reverb.csd].

### Exemple 460. Example of the reverb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o reverb.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; init an audio receiver/mixer
gal init 0

; Instrument #1. (there may be many copies)
instr 1
; generate a source signal
a1 oscili 7000, cpspch(p4), 1
; output the direct sound
out a1
; and add to audio receiver
gal = gal + a1
endin

; (highest instr number executed last)
instr 99
; reverberate whatever is in gal
a3 reverb gal, 1.5
; and output the result
out a3
; empty the receiver for the next pass
gal = 0
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; p4 = frequency (in a pitch-class)
; Play Instrument #1 for a tenth of a second, p4=6.00
i 1 0 0.1 6.00
; Play Instrument #1 for a tenth of a second, p4=6.02
i 1 1 0.1 6.02
; Play Instrument #1 for a tenth of a second, p4=6.04
i 1 2 0.1 6.04
; Play Instrument #1 for a tenth of a second, p4=6.06
i 1 3 0.1 6.06

; Make sure the reverb remains active.
i 99 0 6
e

</CsScore>
</CsoundSynthesizer>

```

## See Also

*alpass, comb, valpass, vcomb*

## Credits

Author: William « Pete » Moss  
 University of Texas at Austin  
 Austin, Texas USA  
 January 2002

# reverb2

reverb2 — Same as the nreverb opcode.

## Description

Same as the *nreverb* opcode.

## Syntax

```
ares reverb2 asig, ktime, khdif [, iskip] [,inumCombs] \  
    [, ifnCombs] [, inumAlpas] [, ifnAlpas]
```

# reverbsc

reverbsc — 8 delay line stereo FDN reverb, based on work by Sean Costello

## Description

8 delay line stereo FDN reverb, with feedback matrix based upon physical modeling scattering junction of 8 lossless waveguides of equal characteristic impedance. Based on Csound orchestra version by Sean Costello.

## Syntax

```
aoutL, aoutR reverbsc ainL, ainR, kfbvl, kfco[, israte[, ipitchm[, iskip]]]
```

## Initialization

*israte* (optional, defaults to the orchestra sample rate) -- assume a sample rate of *israte*. This is normally set to *sr*, but a different setting can be useful for special effects.

*ipitchm* (optional, defaults to 1) -- depth of random variation added to delay times, in the range 0 to 10. The default is 1, but this may be too high and may need to be reduced for held pitches such as piano tones.

*iskip* (optional, defaults to zero) -- if non-zero, initialization of the opcode is skipped, whenever possible.

## Performance

*aoutL*, *aoutR* -- output signals for left and right channel

*ainL*, *ainR* -- left and right channel input. Note that having an input signal on either the left or right channel only will still result in having reverb output on both channels, making this unit more suitable for reverberating stereo input than the *freeverb* opcode.

*kfbvl* -- feedback level, in the range 0 to 1. 0.6 gives a good small "live" room sound, 0.8 a small hall, and 0.9 a large hall. A setting of exactly 1 means infinite length, while higher values will make the opcode unstable.

*kfco* -- cutoff frequency of simple first order lowpass filters in the feedback loop of delay lines, in Hz. Should be in the range 0 to *israte*/2 (not *sr*/2). A lower value means faster decay in the high frequency range.

## Examples

Here is an example of the *reverbsc* opcode. It uses the file *reverbsc.csd* [examples/reverbsc.csd].

### Exemple 461. An example of the reverbsc opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reverb.sc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr      = 48000
ksmps   = 32
nchnls  = 2
0dbfs   = 1

instr 1
a1      vco2 0.85, 440, 10
kfrq    port 100, 0.004, 20000
a1      butterlp a1, kfrq
a2      linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1      = a1 * a2
a2      = a1 * p5
a1      = a1 * p4
denorm  al, a2
aL, aR  reverb.sc a1, a2, 0.85, 12000, sr, 0.5, 1
outs    a1 + aL, a2 + aR
endin

</CsInstruments>
<CsScore>
i 1 0 1 0.71 0.71
i 1 1 1 0 1
i 1 2 1 -0.71 0.71
i 1 3 1 1 0
i 1 4 4 0.71 0.71
e
</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: Istvan Varga  
2005



# rewindscore

rewindscore — Rewinds the playback position of the current score performance.

## Description

Rewinds the playback position of the current score performance..

## Syntax

```
rewindscore
```

## Examples

Here is an example of the rewindscore opcode.

### Exemple 462. Example of the rewindscore opcode.

```
instr 1
  rewindscore
endin
```

## See Also

*setscorepos*

## Credits

Author: Victor Lazzarini  
2008

New in Csound version 5.09

# rezzy

rezzy — A resonant low-pass filter.

## Description

A resonant low-pass filter.

## Syntax

```
ares rezzy asig, xfco, xres [, imode, iskip]
```

## Initialization

*imode* (optional, default=0) -- high-pass or low-pass mode. If zero, *rezzy* is low-pass. If not zero, *rezzy* is high-pass. Default value is 0. (New in Csound version 3.50) *iskip* (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

## Performance

*asig* -- input signal

*xfco* -- filter cut-off frequency in Hz. As of version 3.50, may i-,k-, or a-rate.

*xres* -- amount of resonance. Values of 1 to 100 are typical. Resonance should be one or greater. As of version 3.50, may a-rate, i-rate, or k-rate.

*rezzy* is a resonant low-pass filter created empirically by Hans Mikelson.

## Examples

Here is an example of the *rezzy* opcode. It uses the file *rezzy.csd* [examples/rezzy.csd].

### Exemple 463. Example of the rezzy opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rezzy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
; Use a nice sawtooth waveform.
asig vco 32000, 220, 1

; Vary the filter-cutoff frequency from .2 to 2 KHz.
kfco line 200, p3, 2000

; Set the resonance amount.
kres init 25

al rezzzy asig, kfco, kres

out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vco opcode.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*biquad, moogvcf*

## Credits

Author: Hans Mikelson  
October 1998

Example written by Kevin Conder.

New in Csound version 3.49

# rigoto

rigoto — Transfère le contrôle durant une phase de réinitialisation.

## Description

Semblable à *igoto*, mais n'agit que dans une phase de *réinitialisation* (*reinit*) (c'est-à-dire qu'il n'opère pas pendant l'initialisation standard). Cette instruction est utile pour ignorer les unités qui ne doivent pas être réinitialisées.

## Syntaxe

```
rigoto label
```

## Voir Aussi

*cigoto, igoto, reinit, rireturn*

# rireturn

rireturn — Termine une phase de réinitialisation.

## Description

Termine une phase de *réinitialisation* (*reinit*) (c'est-à-dire qu'il n'opère pas pendant l'initialisation standard). Cette instruction, ou un *endin*, provoquera la reprise de l'exécution normale.

## Syntaxe

`rireturn`

## Exemples

Les instructions suivantes génèrent un signal de contrôle exponentiel dont les valeurs vont de 440 à 880 exactement dix fois pendant la durée p3. Elles utilisent le fichier *reinit.csd* [exemples/reinit.csd].

### Exemple 464. Exemple de l'opcode rireturn.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o reinit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1

reset:
    timeout 0, p3/10, contin
    reinit reset

contin:
    kLine expon 440, p3/10, 880
    aSig oscil 10000, kLine, 1
    out aSig
    rireturn

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1

i1 0 10
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*reinit, rigoto*

# rms

rms — Détermine la valeur efficace d'un signal audio.

## Description

Détermine la valeur efficace d'un signal audio. La valeur instantanée passe à travers un filtre passe-bas pour en sortir une valeur moyenne comme dans un VU-mètre.

## Syntaxe

```
kres rms asig [, ihp] [, iskip]
```

## Initialisation

*ihp* (facultatif, 10 par défaut) -- point à mi-puissance (en Hz) d'un d'un filtre passe-bas interne spécial. La valeur par défaut est 10.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace de données interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio en entrée

*kres* -- valeur efficace du signal d'entrée issue du filtre passe-bas

Les valeurs de sortie *kres* de *rms* suivent la valeur efficace de l'entrée audio *asig*. Cette unité n'est pas un modificateur de signal, mais fonctionne plutôt comme une mesure de la puissance du signal. Elle utilise un filtre passe-bas interne pour rendre la réponse plus lisse. On peut utiliser *ihp* pour contrôler ce lissage. Plus les valeurs sont importantes, plus la mesure est "dynamique".

On peut aussi utiliser cet opcode comme suiveur d'enveloppe.

La sortie *kres* de cet opcode est donnée en amplitude et dépend de *odbfs*. Pour une sortie en décibels, il faut utiliser *dbamp*

## Exemples

```
arms rms    asig ; get rms value of signal asig
```

Voici un exemple de l'opcode rms. Il utilise le fichier *rms.csd* [examples/rms.csd].

### Exemple 465. Exemple de l'opcode rms.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur

l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d  -m0      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rms.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 128
nchnls = 1

;Example by Andres Cabrera 2007

0dbfs = 1

FLpanel "rms", 400, 100, 50, 50
    gkrms text, gihrms text FLtext "Rms", -100, 0, 0.1, 3, 110, 30, 60, 50
    gkihp, gihandle FLtext "ihp", 0, 10, 0.05, 1, 100, 30, 220, 50
    gkrms slider, gihrms slider FLslider "", -60, -0.5, -1, 5, -1, 380, 20, 10, 10

FLpanelEnd
FLrun

FLsetVal_i 5, gihandle
; Instrument #1.
instr 1
    al inch 1

label:
    kval rms al, i(gkihp) ;measures rms of input channel 1
rreturn

    kval = dbamp(kval) ; convert to db full scale
    printk 0.5, kval
    FLsetVal 1, kval, gihrms slider ;update the slider and text values
    FLsetVal 1, kval, gihrms text
    knewihp changed gkihp ; reinit when ihp text has changed
    if (knewihp == 1) then
        reinit label ;needed because ihp is an i-rate parameter
    endif
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute
i 1 0 60
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*balance, gain*



# rnd

rnd — Retourne un nombre aléatoire dans un intervalle unipolaire au taux de l'argument.

## Description

Retourne un nombre aléatoire dans un intervalle unipolaire au taux de l'argument.

## Syntaxe

`rnd(x)` (taux-i ou -k seulement)

Où l'argument entre parenthèses peut être une expression. Ces convertisseurs de valeur échantillonnent une séquence aléatoire globale, mais sans référencer une *racine*. Le résultat peut devenir un terme d'une expression ultérieure.

## Exécution

Retourne un nombre aléatoire dans l'intervalle unipolaire allant de 0 à  $x$ .

## Exemples

Voici un exemple de l'opcode rnd. Il utilise le fichier *rnd.csd* [examples/rnd.csd].

### Exemple 466. Exemple de l'opcode rnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a random number from 0 to 1.
il = rnd(1)
print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #1 for one second.
```

```
i 1 1 1  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
rnd at i-rate: 0.973500   rnd at k-rate: 0.139405  
rnd at i-rate: 0.973500   rnd at k-rate: 0.040065  
rnd at i-rate: 0.973500   rnd at k-rate: 0.412845  
rnd at i-rate: 0.973500   rnd at k-rate: 0.440650  
rnd at i-rate: 0.973500   rnd at k-rate: 0.663581  
rnd at i-rate: 0.973500   rnd at k-rate: 0.876723  
rnd at i-rate: 0.973500   rnd at k-rate: 0.302459  
rnd at i-rate: 0.973500   rnd at k-rate: 0.398580  
rnd at i-rate: 0.973500   rnd at k-rate: 0.448875  
rnd at i-rate: 0.973500   rnd at k-rate: 0.907728
```

## Voir Aussi

*birnd*

## Crédits

Auteur: Barry L. Vercoe  
MIT  
Cambridge, Massachussetts  
1997

Exemple original écrit par Kevin Conder. Modifié par John Harrison.

# rnd31

rnd31 — Opcodes aléatoires bipolaires sur 31 bit avec une distribution contrôlée.

## Description

Opcodes aléatoires bipolaires sur 31 bit avec une distribution contrôlée. Ces unités sont portables, c-à-d qu'avec la même valeur de graine on obtiendra la même séquence aléatoire sur tous les systèmes. La distribution des nombres aléatoires générés peut être changée au taux-k.

## Syntaxe

ax **rnd31** kscl, krpow [, iseed]

ix **rnd31** iscl, irpow [, iseed]

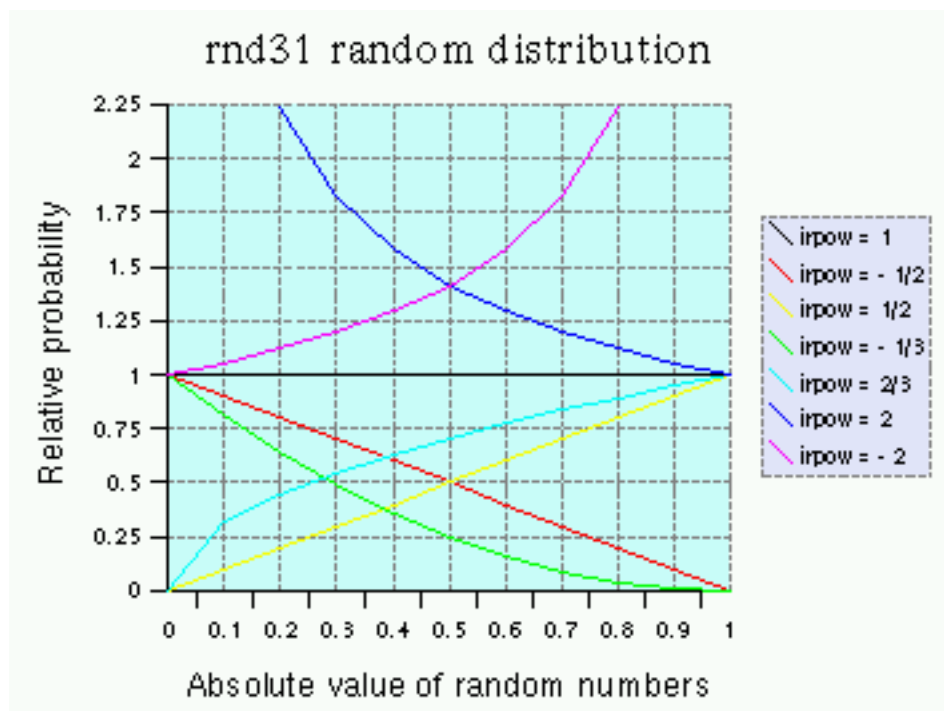
kx **rnd31** kscl, krpow [, iseed]

## Initialisation

*ix* -- valeur de sortie au taux-i.

*iscl* -- mise à l'échelle de la sortie. Les nombres aléatoires générés sont compris entre *-iscl* et *iscl*.

*irpow* -- contrôle la distribution des nombres aléatoires. Si *irpow* est positif, la distribution aléatoire (*x* compris entre -1 et 1) est  $abs(x)^{(1/irpow) - 1}$  ; pour des valeurs négatives de *irpow*, elle vaut  $(1 - abs(x))^{((-1/irpow) - 1)}$ . En fixant *irpow* à -1, 0 ou 1 on obtiendra une distribution uniforme (c'est aussi plus rapide à calculer).



Un graphique des distributions pour différentes valeurs de *irpow*.

*iseed* (facultatif, par défaut=0) -- valeur de la graine pour le générateur de nombres aléatoires (nombre entier positif compris entre 1 et 2147483646 ( $2^{31} - 2$ )). Avec une valeur nulle ou négative la graine est prise à partir de l'horloge du système (c'est le comportement par défaut). Une graine à partir de l'horloge du système nous garantit la génération de séquences aléatoires différentes, même si plusieurs opcodes aléatoires sont appelés dans un temps très court.

Dans les versions de *taux-a* et de *taux-k* la graine est fixée à l'initialisation de l'opcode. Avec une sortie de *taux-i*, si la graine est nulle ou négative, elle sera prise à partir de l'horloge du système lors du premier appel, puis retournera la valeur suivante de la séquence aléatoire lors des appels successifs ; les valeurs positives de la graine sont fixées à tous les appels de *taux-i*. La graine est locale pour les unités de *taux-a* et *-k*, et globale pour les unités de *taux-i*.



## Notes

- bien que des valeurs de graines allant jusqu'à 2147483646 soient permises, il est recommandé d'utiliser des nombres plus petits ( $< 1000000$ ) pour des raisons de portabilité, car les grands nombres peuvent être arrondis à une valeur différente si l'on utilise des nombres flottants sur 32 bit.
- *rnd31* au *taux-i* avec une graine positive produira toujours la même valeur en sortie (ce n'est pas un bogue). Pour obtenir des valeurs différentes, fixer la graine à 0 dans les appels successifs, ce qui retournera la valeur suivante de la séquence aléatoire.

## Exécution

*ax* -- valeur de sortie au *taux-a*.

*kx* -- valeur de sortie au *taux-k*.

*kscl* -- mise à l'échelle de la sortie. Les nombres aléatoires générés sont compris entre *-kscl* et *kscl*. Semblable à *iscl*, mais il peut être modifié au *taux-k*.

*krpow* -- contrôle la distribution des nombres aléatoires. Semblable à *irpow*, mais il peut être modifié au *taux-k*.

## Exemples

Voici un exemple de l'opcode *rnd31*. Il utilise le fichier *rnd31.csd* [examples/rnd31.csd].

### Exemple 467. Exemple de l'opcode *rnd31* au *taux-a*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create random numbers at a-rate in the range -2 to 2 with
; a triangular distribution, seed from the current time.
a31 rnd31 2, -0.5

; Use the random numbers to choose a frequency.
afreq = a31 * 500 + 100

a1 oscil 30000, afreq, 1
out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode `rnd31` au taux-k. Il utilise le fichier `rnd31_krate.csd` [exemples/rnd31\_krate.csd].

### Exemple 468. Exemple de l'opcode `rnd31` au taux-k.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o rnd31_krate.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create random numbers at k-rate in the range -1 to 1
; with a uniform distribution, seed=10.
k1 rnd31 1, 0, 10

printks "k1=%f\\n", 0.1, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

```

```
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1= 0.112106
k1=-0.274665
k1= 0.403933
```

Here is an example of the `rnd31` opcode that uses the number 7 as a seed value. It uses the file `rnd31_seed7.csd` [examples/rnd31\_seed7.csd].

**Exemple 469.** An example of the `rnd31` opcode that uses the number 7 as a seed value.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31_seed7.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; i-rate random numbers with linear distribution, seed=7.
; (Note that the seed was used only in the first call.)
i1 rnd31 1, 0.5, 7
i2 rnd31 1, 0.5
i3 rnd31 1, 0.5

print i1
print i2
print i3
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1: i1 = -0.649
instr 1: i2 = -0.761
instr 1: i3 = 0.677
```

Voici un exemple de l'opcode `rnd31` qui utilise l'horloge du système comme graine. Il utilise le fichier `rnd31_time.csd` [examples/rnd31\_time.csd].

### Exemple 470. Exemple de l'opcode `rnd31` qui utilise l'horloge du système comme graine.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31_time.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; i-rate random numbers with linear distribution,
; seeding from the current time. (Note that the seed
; was used only in the first call.)
i1 rnd31 1, 0.5, 0
i2 rnd31 1, 0.5
i3 rnd31 1, 0.5

print i1
print i2
print i3
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  i1 = -0.691
instr 1:  i2 = -0.686
instr 1:  i3 = -0.358
```

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.16

# round

**round** — Retourne la valeur entière la plus proche de  $x$  ; si la partie décimale de  $x$  vaut exactement 0.5, la direction de l'arrondi est indéfinie.

## Description

La valeur entière la plus proche de  $x$  ; si la partie décimale de  $x$  vaut exactement 0.5, la direction de l'arrondi est indéfinie.

## Syntaxe

**round**( $x$ ) (des arguments de `taux-i`, `-k` ou `-a` sont permis)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur réalisent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Voir Aussi

*abs, exp, int, log, log10, i, sqrt*

## Crédits

Auteur : Istvan Varga  
Nouveau dans Csound 5  
2005



# rspline

rspline — Génère des courbes splines aléatoires.

## Description

Génère des courbes splines aléatoires.

## Syntaxe

```
ares rspline xrangeMin, xrangeMax, kcpsMin, kcpsMax
```

```
kres rspline krangeMin, krangeMax, kcpsMin, kcpsMax
```

## Exécution

*kres, ares* -- Signal de sortie.

*xrangeMin, xrangeMax* -- Intervalle des valeurs des points générés aléatoirement.

*kcpsMin, kcpsMax* -- Intervalle de définition du taux de génération des points. Les limites minimale et maximale sont exprimées en Hz.

*rspline* (générateur de courbe spline aléatoire) est semblable à *jspline* mais l'intervalle de sortie est défini par deux valeurs limites. De plus, ici, l'intervalle de sortie réel pourra légèrement dépasser les valeurs données à cause des courbes d'interpolation entre chaque paire de points aléatoires.

Actuellement les courbes générées sont assez lisses quand *cspMin* n'est pas trop différent de *cpsMax*. Quand l'intervalle *cpsMin-cpsMax* est grand, quelques petites discontinuités peuvent se produire, mais, dans la plupart des cas, cela ne devrait pas poser de problème. L'algorithme sera peut-être amélioré dans les prochaines versions.

Ces opcodes sont souvent meilleurs que *jitter* lorsque l'on veut un rendu « naturel » ou « analogique » de sons numériques. On peut aussi les utiliser dans la composition algorithmique, pour générer des lignes mélodiques aléatoires lisses lors d'une utilisation conjointe avec l'opcode *samphold*.

Noter que le résultat est assez différent de celui que l'on obtiendrait en filtrant un bruit blanc, et que l'on peut ainsi obtenir un contrôle bien plus précis.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.15

# rtclock

rtclock — Lit l'horloge temps réel du système d'exploitation.

## Description

Lit l'horloge temps réel du système d'exploitation.

## Syntaxe

```
ires rtclock
```

```
kres rtclock
```

## Exécution

Lit l'horloge temps réel du système d'exploitation. Sous Windows, celle-ci ne change qu'une fois par seconde. Sous GNU/Linux, elle change chaque microseconde. Le comportement sous les autres systèmes varie.

## Exemples

Voici un exemple de l'opcode `rtclock`. Il utilise le fichier `rtclock.csd` [examples/rtclock.csd].

### Exemple 471. Exemple de l'opcode `rtclock`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rtclock.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1
instr 1
; Get the system time.
k1 rtclock
; Print it once per second.
printk 1, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
```

```
i 1 0 2  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
i 1 time 0.00002: 1018236096.00000  
i 1 time 1.00002: 1018236224.00000
```

## Crédits

Auteur : John ffitch

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.10

# s16b14

s16b14 — Creates a bank of 16 different 14-bit MIDI control message numbers.

## Description

Creates a bank of 16 different 14-bit MIDI control message numbers.

## Syntax

```
i1,...,i16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \  
    initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16, ifn16  
  
k1,...,k16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \  
    initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16, ifn16
```

## Initialization

*i1 ... i64* -- output values

*ichan* -- MIDI channel (1-16)

*ictlno\_msb1 .... ictlno\_msb32* -- MIDI control number, most significant byte (0-127)

*ictlno\_lsb1 .... ictlno\_lsb32* -- MIDI control number, least significant byte (0-127)

*imin1 ... imin64* -- minimum values for each controller

*imax1 ... imax64* -- maximum values for each controller

*init1 ... init64* -- initial value for each controller

*ifn1 ... ifn64* -- function table for conversion for each controller

*icutoff1 ... icutoff64* -- low-pass filter cutoff frequency for each controller

## Performance

*k1 ... k64* -- output values

*s16b14* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*s16b14* allows a bank of 16 different MIDI control message numbers. It uses 14-bit values instead of MIDI's normal 7-bit values.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *sl6b14*, there is not an initial value input argument. The output is taken directly from the current status of internal controller array of Csound.

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# s32b14

s32b14 — Creates a bank of 32 different 14-bit MIDI control message numbers.

## Description

Creates a bank of 32 different 14-bit MIDI control message numbers.

## Syntax

```
i1,...,i32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \  
    initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32, ifn32  
  
k1,...,k32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \  
    initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32, ifn32
```

## Initialization

*i1 ... i64* -- output values

*ichan* -- MIDI channel (1-16)

*ictlno\_msb1 .... ictlno\_msb32* -- MIDI control number, most significant byte (0-127)

*ictlno\_lsb1 .... ictlno\_lsb32* -- MIDI control number, least significant byte (0-127)

*imin1 ... imin64* -- minimum values for each controller

*imax1 ... imax64* -- maximum values for each controller

*init1 ... init64* -- initial value for each controller

*ifn1 ... ifn64* -- function table for conversion for each controller

*icutoff1 ... icutoff64* -- low-pass filter cutoff frequency for each controller

## Performance

*k1 ... k64* -- output values

*s32b14* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*s32b14* allows a bank of 32 different MIDI control message numbers. It uses 14-bit values instead of MIDI's normal 7-bit values.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *s32b14*, there is not an initial value input argument. The output is taken directly from the current status of internal controller array of Csound.

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# scale

scale — Signal de pondération arbitraire.

## Description

Met les valeurs entrantes à l'échelle d'un intervalle défini par l'utilisateur. Semblable à l'objet de pondération que l'on trouve dans les langages de flux de données les plus connus.

## Syntaxe

```
kscl scale kinput, kmax, kmin
```

## Exécution

*kin* -- Valeur d'entrée. Elle peut provenir de n'importe quelle source au taux-k pourvu que la sortie de cette dernière soit comprise entre 0 et 1.

*kmin* -- Valeur minimale de l'intervalle de pondération.

*kmax* -- Valeur maximale de l'intervalle de pondération.

## Exemples

Voici un exemple de l'opcode scale. Il utilise le fichier *scale.csd* [examples/scale.csd].

### Exemple 472. Exemple de l'opcode scale.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent
-odac      -iadc      -d      ;;realtime output
</CsOptions>
<CsInstruments>

sr = 22050
ksmps = 10
nchnls = 2

/*--- */

instr 1 ; scale test

kmod ctrl1 1, 1, 0, 1
printk2 kmod

kout scale kmod, 0, -127
printk2 kout

endin

/*--- */
</CsInstruments>
<CsScore>
```



```
i1 0 8888
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*gainslider, logcurve, expcurve*

## Crédits

Auteur : David Akbari  
Octobre  
2006

# samphold

samphold — Performs a sample-and-hold operation on its input.

## Description

Performs a sample-and-hold operation on its input.

## Syntax

```
ares samphold asig, agate [, ival] [, ivstor]
```

```
kres samphold ksig, kgate [, ival] [, ivstor]
```

## Initialization

*ival*, *ivstor* (optional) -- controls initial disposition of internal save space. If *ivstor* is zero the internal « hold » value is set to *ival* ; else it retains its previous value. Defaults are 0,0 (i.e. init to zero)

## Performance

*kgate*, *xgate* -- controls whether to hold the signal.

*samphold* performs a sample-and-hold operation on its input according to the value of *gate*. If *gate* != 0, the input samples are passed to the output; If *gate* = 0, the last output value is repeated. The controlling *gate* can be a constant, a control signal, or an audio signal.

## Examples

```
asrc buzz      10000,440,20, 1      ; band-limited pulse train
adif diff      asrc                ; emphasize the highs
anew balance   adif, asrc           ; but retain the power
agate reson    asrc,0,440           ; use a lowpass of the original
asamp samphold anew, agate          ; to gate the new audiosig
aout tone      asamp,100            ; smooth out the rough edges
```

## See Also

*diff*, *downsamp*, *integ*, *interp*, *upsamp*

# sandpaper

sandpaper — Modèle semi-physique d'un son de papier de verre.

## Description

*sandpaper* est un modèle semi-physique d'un son de papier de verre. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares sandpaper iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 128.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,998 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,999 ce qui signifie que la valeur par défaut de *idamp* est 0,5. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode sandpaper. Il utilise le fichier *sandpaper.csd* [examples/sandpaper.csd].

### Exemple 473. Exemple de l'opcode sandpaper.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o sandpaper.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra -----

    sr =          44100
    kr =          4410
    ksmpr =        10
    nchnls =        1

instr 01                                ;an example of sandpaper blocks
a1      line 2, p3, 2                    ;preset amplitude increase
a2      sandpaper p4, 0.01                ;sandpaper needs a little amp help at these settings
a3      product a1, a2                    ;increase amplitude
        out a3
        endin

</CsInstruments>
<CsScore>

;score -----

i1 0 1 26000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*cabasa, crunch, sekere, stix*

## Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# scanhammer

scanhammer — Copie d'une table vers une autre avec contrôle du gain.

## Description

C'est une variante de *tablecopy*, qui copie d'une table vers une autre, à partir de *ipos*, et avec un contrôle du gain. Le nombre de points copiés est déterminé par la longueur de la source. Les autres points ne sont pas changés. On peut utiliser cet opcode pour « frapper » une corde dans le code de synthèse par balayage.

## Syntaxe

```
scanhammer isrc, idst, ipos, imult
```

## Initialisation

*isrc* -- table de fonction source.

*idst* -- table de fonction destination.

*ipos* -- position de départ (en points).

*imult* -- multiplicateur du gain. S'il vaut 0, les valeurs ne seront pas modifiées.

## Voir Aussi

*scantable*

## Crédits

Auteur : Matt Gilliard  
Avril 2002

Nouveau dans la version 4.20

## scans

scans — Génère une sortie audio au moyen de la synthèse par balayage.

## Description

Génère une sortie audio au moyen de la synthèse par balayage.

## Syntaxe

```
ares scans kamp, kfreq, ifn, id [, iorder]
```

## Initialisation

*ifn* -- ftable contenant la trajectoire du balayage. C'est une série de nombres qui contiennent les adresses des masses. L'ordre de ces adresses est utilisé comme chemin de balayage. Ne doit pas contenir de valeurs supérieures au nombre de masses, ou des nombres négatifs. Voir l'*introduction à la section sur la synthèse par balayage*.

*id* -- numéro d'ID de la forme d'onde de l'opcode *scanu* à utiliser.

*iorder* (facultatif, 0 par défaut) -- ordre de l'interpolation utilisée en interne. Peut prendre n'importe quelle valeur comprise entre 1 et 4, et vaut 4 par défaut, qui est l'interpolation quartique. 2 est l'interpolation quadratique et 1 l'interpolation linéaire. Les nombres les plus élevés donnent un traitement plus lent, mais pas nécessairement meilleur.

## Exécution

*kamp* -- amplitude de la sortie. Noter que l'amplitude résultante dépend aussi des valeurs instantanées de la table d'onde. Ce nombre est en fait la facteur de pondération de la table d'onde.

*kfreq* -- fréquence de balayage

## Exemples

Voici un exemple de synthèse par balayage. Il utilise les fichiers *scans.csd* [examples/scans.csd], et *string-128.matrix* [examples/string-128.matrix].

### Exemple 474. Exemple de l'opcode scans.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o scans.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 128
nchnls = 1

instr 1
a0 = 0
; scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, kstif, kcentr, kdamp, ileft,
scanu 1, .01, 6, 2, 3, 4, 5, 2, .1, .1, -.01, .1,
;ar scans kamp, kfreq, ifntraj, id
a1 scans ampdb(p4), cspch(p5), 7, 2
out a1
endin

</CsInstruments>
<CsScore>

; Initial condition
f1 0 128 7 0 64 1 64 0

; Masses
f2 0 128 -7 1 128 1

; Spring matrices
f3 0 16384 -23 "string-128.matrix"

; Centering force
f4 0 128 -7 0 128 2

; Damping
f5 0 128 -7 1 128 1

; Initial velocity
f6 0 128 -7 0 128 0

; Trajectories
f7 0 128 -5 .001 128 128

; Note list
i1 0 10 86 6.00
i1 11 14 86 7.00
i1 15 20 86 5.00
e

</CsScore>
</CsoundSynthesizer>
```

Le fichier de la matrice « string-128.matrix », ainsi que d'autres matrices, est aussi disponible dans un *fichier zip* [<http://www.csounds.com/scanned/zip/scanmatrices.zip>] depuis la *page Scanned Synthesis* [<http://www.csounds.com/scanned/>] à cSounds.com.

## Crédits

Auteur : Paris Smaragdis  
MIT Media Lab  
Boston, Massachussetts USA

Nouveau dans la version 4.05 de Csound

# scantable

scantable — Une implémentation simplifiée de la synthèse par balayage.

## Description

Une implémentation simplifiée de la synthèse par balayage. C'est l'implémentation d'une corde circulaire parcourue au moyen de tables externes. Cet opcode permet la modification directe et la lecture des valeurs avec les opcodes de table.

## Syntaxe

```
aout scantable kamp, kpch, ipos, imass, istiff, idamp, ivel
```

## Initialisation

*ipos* -- table contenant le tableau de position.

*imass* -- table contenant la masse de la corde.

*istiff* -- table contenant la raideur de la corde.

*idamp* -- table contenant les facteurs d'atténuation de la corde.

*ivel* -- table contenant les vitesses.

## Exécution

*kamp* -- amplitude (gain) de la corde.

*kpch* -- la fréquence de balayage de la corde.

## Exemples

Voici un exemple de l'opcode scantable. Il utilise le fichier *scantable.csd* [examples/scantable.csd].

### Exemple 475. Exemple de l'opcode scantable.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o scantable.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
```



```
ksmps = 10
nchnls = 1

; Table #1 - initial position
git1 ftgen 1, 0, 128, 7, 0, 64, 1, 64, 0
; Table #2 - masses
git2 ftgen 2, 0, 128, -7, 1, 128, 1
; Table #3 - stiffness
git3 ftgen 3, 0, 128, -7, 0, 64, 100, 64, 0
; Table #4 - damping
git4 ftgen 4, 0, 128, -7, 1, 128, 1
; Table #5 - initial velocity
git5 ftgen 5, 0, 128, -7, 0, 128, 0

; Instrument #1.
instr 1
  kamp init 20000
  kpch init 220
  ipos = 1
  imass = 2
  istiff = 3
  idamp = 4
  ivel = 5

  a1 scantable kamp, kpch, ipos, imass, istiff, idamp, ivel
  a2 dcblock a1

  out a2
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for ten seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*scanhammer*

## Crédits

Auteur : Matt Gilliard  
Avril 2002

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.20

# scanu

scanu — Calcule la forme d'onde et la table d'onde à utiliser dans la synthèse par balayage.

## Description

Calcule la forme d'onde et la table d'onde à utiliser dans la synthèse par balayage.

## Syntaxe

```
scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \  
      kstif, kcentr, kdamp, ileft, iright, kpos, kstrngth, ain, idisp, id
```

## Initialisation

*init* -- la position initiale des masses. Si c'est un nombre négatif, alors la valeur absolue de *init* indique la table à utiliser pour la forme du marteau. Si *init* > 0, il représente le nombre de masses attendu.

*ifnvel* -- ftable contenant la vitesse initiale de chaque masse. Sa taille est le nombre de masses attendu.

*ifnmass* -- ftable contenant la valeur de chaque masse. Sa taille est le nombre de masses attendu.

*ifnstif* -- ftable contenant la raideur du ressort de chaque connexion. Sa taille est le carré du nombre de masses attendu. Ses données sont ordonnées selon la succession des lignes de la matrice de connexion du système.

*ifncentr* -- ftable contenant la force de centrage de chaque masse. Sa taille est le nombre de masses attendu.

*ifndamp* -- ftable contenant le facteur d'amortissement de chaque masse. Sa taille est le nombre de masses attendu.

*ileft* -- si *init* < 0, position du marteau de gauche (*ileft* = 0 frappe complètement à gauche, *ileft* = 1 frappe complètement à droite).

*iright* -- si *init* < 0, position du marteau de droite (*iright* = 0 frappe complètement à gauche, *iright* = 1 frappe complètement à droite).

*idisp* -- s'il vaut 0, il n'y a pas d'affichage des masses.

*id* -- s'il est positif, c'est l'ID de l'opcode. Il est utilisé pour relier l'opcode de balayage au bon générateur de forme d'onde. S'il est négatif, sa valeur absolue indique la table d'onde dans laquelle sera écrite la forme d'onde. Cette forme d'onde peut être utilisée par la suite par un autre opcode pour générer du son. Le contenu initial de cette table sera écrasé.

## Exécution

*kmass* -- pondère les masses

*kstif* -- pondère la raideur des ressorts

*kcentr* -- pondère la force de centrage

*kdamp* -- pondère l'amortissement

*kpos* -- position d'un marteau actif le long de la corde (*kpos* = 0 est complètement à gauche, *kpos* = 1 est complètement à droite). La forme du marteau est déterminée par *init* et sa puissance de percussion est *kstrngth*.

*kstrngth* -- puissance utilisée par le marteau actif

*ain* -- entrée audio qui s'ajoute à la vitesse des masses. L'amplitude ne doit pas être trop grande.

## Exemples

Pour un exemple, voir la documentation de *scans*.

## Crédits

Auteur : Paris Smaragdis  
MIT Media Lab  
Boston, Massachusetts USA  
Mars 2000

Nouveau dans la version 4.05 de Csound

# scoreline

scoreline — Délivre un ou plusieurs évènements de ligne de partition depuis un instrument.

## Description

*scoreline* délivre un ou plusieurs évènements de partition, si *ktrig* vaut 1, à chaque période *k*. Il peut gérer les chaînes de caractères dans les mêmes conditions que dans la partition standard. Les chaînes de caractères sur plusieurs lignes sont acceptées, en utilisant `{ { }` pour encadrer la chaîne de caractères.

## Syntaxe

```
scoreline Sin, ktrig
```

## Initialisation

« *Sin* » -- une chaîne de caractères (entre guillemets ou encadrée par `{ { }`), contenant un ou plusieurs évènements de partition.

## Exécution

« *ktrig* » -- déclencheur d'évènement, 1 délivre l'évènement de partition, 0 l'ignore.

## Exemples

Voici un exemple de l'opcode *scoreline*.

### Exemple 476. Exemple

```
instr 1
  ktrig init 1
  scoreline {{
    i 2 0 3 "flutec3.wav"
    i 2 1 3 "clarac3.wav"
  }}, ktrig
  ktrig = 0
endin

instr 2
  aout soundin p4
  out aout
endin
```

Vous pouvez utiliser des opcodes de chaîne de caractères comme *sprintfk* pour produire les chaînes de caractères à passer à *scoreline* comme ceci :

```
Sfil = "/Volumes/Bla/file.aif"
String sprintfk {{i 2 0 %f "%s" %f %f %f %f}}, idur, Sfil, p5, p6, knorm, iskip
scoreline String, ktrig
```

## Voir Aussi

*event, event\_i, schedule, schedwhen, schedkwhen, schedkwhennamed, scoreline\_i*

## Crédits

Auteur : Victor Lazzarini, 2007

# scoreline\_i

`scoreline_i` — Délivre un ou plusieurs évènements de ligne de partition depuis un instrument pendant la phase d'initialisation.

## Description

`scoreline_i` délivre des évènements de partition au cours de la phase d'initialisation. Il peut gérer les chaînes de caractères dans les mêmes conditions que dans la partition standard. Les chaînes de caractères sur plusieurs lignes sont acceptées, en utilisant `{{ }}` pour encadrer la chaîne de caractères.

## Syntaxe

```
scoreline_i Sin
```

## Initialisation

« *Sin* » -- une chaîne de caractères (entre guillemets ou encadrée par `{{ }}`), contenant un ou plusieurs évènements de partition.

## Exemples

Voici en exemple de l'opcode `scoreline_i`.

### Exemple 477. Exemple

```
instr 1
  scoreline_i {{
    i 2 0 3 "flutec3.wav"
    i 2 1 3 "clar3.wav"
  }}
endin
instr 2
  aout soundin p4
  out aout
endin
```

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

## Voir Aussi

*event*, *event\_i*, *schedule*, *schedwhen*, *schedkwhen*, *schedkwhennamed*, *scoreline*

## Crédits

Auteur : Victor Lazzarini, 2007

# schedkwhen

**schedkwhen** — Ajoute un nouvel évènement de partition généré par un signal de déclenchement de taux-k.

## Description

Ajoute un nouvel évènement de partition généré par un signal de déclenchement de taux-k.

## Syntaxe

```
schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur \  
[, ip4] [, ip5] [...]
```

```
schedkwhen ktrigger, kmintim, kmaxnum, "insname", kwhen, kdur \  
[, ip4] [, ip5] [...]
```

## Initialisation

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*ip4*, *ip5*, ... -- Equivalent à *p4*, *p5*, etc., dans une *instruction i* de partition.

## Exécution

*ktrigger* -- déclenche un nouvel évènement de partition. Si *ktrigger* = 0, aucun nouvel évènement n'est déclenché.

*kmintim* -- intervalle de temps minimum entre les évènements générés, en secondes. Si *kmintim* <= 0, il n'y a aucune limite de temps. Si *kinsnum* est négatif (pour arrêter un instrument), ce test est ignoré.

*kmaxnum* -- nombre maximum d'instances simultanées de l'instrument *kinsnum* autorisées. Si le nombre d'instances existantes de *kinsnum* est >= *kmaxnum*, aucun nouvel évènement n'est généré. Si *kmaxnum* est <= 0, il n'est pas utilisé pour limiter la génération d'évènement. Si *kinsnum* est négatif (pour arrêter un instrument), ce test est ignoré.

*kinsnum* -- numéro d'un instrument. Equivalent à *p1* dans une *instruction i* de partition.

*kwhen* -- date de début du nouvel évènement. Equivalent à *p2* dans une *instruction i* de partition. Mesurée à partir de l'instant de l'évènement déclencheur. *kwhen* doit être >= 0. Si *kwhen* > 0, l'instrument ne sera pas initialisé jusqu'à ce que cette date soit atteinte.

*kdur* -- durée de l'évènement. Equivalent à *p3* dans une *instruction i* de partition. Si *kdur* = 0, l'instrument ne fera qu'une phase d'initialisation, sans exécution. Si *kdur* est négatif, une note tenue est démarrée. (Voir *ihold* et *instruction i*.)



### Note

Dans l'attente d'évènements à déclencher par *schedkwhen*, l'exécution doit continuer, ou Csound pourrait se terminer si aucun évènement de partition n'est attendu. Pour garantir une exécution continue, on peut utiliser une *instruction f0* dans la partition.





## Note

Noter que l'opcode *schedkwhen* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici une exemple de l'opcode *schedkwhen*. Il utilise le fichier *schedkwhen.csd* [examples/schedkwhen.csd].

### Exemple 478. Exemple de l'opcode *schedkwhen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o schedkwhen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
; Use the fourth p-field as the trigger.
ktrigger = p4
kmintim = 0
kmaxnum = 2
kinsnum = 2
kwhen = 0
kdur = 0.5

; Play Instrument #2 at the same time, if the trigger is set.
schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur

; Play a high note.
a1 oscils 10000, 880, 1
out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
; Play a low note.
a1 oscils 10000, 220, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = trigger for Instrument #2 (when p4 > 0).
; Play Instrument #1 for half a second, no trigger.
i 1 0 0.5 0
; Play Instrument #1 for half a second, trigger Instrument #2.
```

```
i 1 1 0.5 1  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*event, event\_i, schedule, schedwhen, schedkwhennamed, scoreline, scoreline\_i*

## Crédits

Auteur : Rasmus Ekman  
EMS, Stockholm, Suède

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.59 de Csound

# schedkwhennamed

`schedkwhennamed` — Semblable à `schedkwhen` mais avec un instrument nommé dans la phase d'initialisation.

## Description

Semblable à *schedkwhen* mais avec un instrument nommé dans la phase d'initialisation.

## Syntaxe

```
schedkwhennamed ktrigger, kmintim, kmaxnum, "name", kwhen, kdur \
[, ip4] [, ip5] [...]
```

## Initialisation

*ip4*, *ip5*, ... -- Equivalent à *p4*, *p5*, etc., dans une *instruction i* de partition.

## Exécution

*ktrigger* -- déclenche un nouvel évènement de partition. Si *ktrigger* = 0, aucun nouvel évènement n'est déclenché.

*kmintim* -- intervalle de temps minimum entre les évènements générés, en secondes. Si *kmintim* est inférieur ou égal à 0, il n'y a aucune limite de temps.

*kmaxnum* -- nombre maximum d'instances simultanées de l'instrument nommé autorisées. Si le nombre d'instances existantes de l'instrument nommé est supérieur ou égal à *kmaxnum*, aucun nouvel évènement n'est généré. Si *kmaxnum* est inférieur ou égal à 0, il n'est pas utilisé pour limiter la génération d'évènement.

"*name*" -- le nom de l'instrument.

*kwhen* -- date de début du nouvel évènement. Equivalent à *p2* dans une *instruction i* de partition. Mesurée à partir de l'instant de l'évènement déclencheur. *kwhen* doit être supérieure ou égale à 0. Si *kwhen* est supérieure à 0, l'instrument ne sera pas initialisé jusqu'à ce que cette date soit atteinte.

*kdur* -- durée de l'évènement. Equivalent à *p3* dans une *instruction i* de partition. Si *kdur* vaut 0, l'instrument ne fera qu'une phase d'initialisation, sans exécution. Si *kdur* est négatif, une note tenue est démarrée. (Voir *ihold* et *instruction i*.)



### Note

Dans l'attente d'évènements à déclencher par *schedkwhennamed*, l'exécution doit continuer, ou Csound pourrait se terminer si aucun évènement de partition n'est attendu. Pour garantir une exécution continue, on peut utiliser une *instruction f0* dans la partition.



### Note

Noter que l'opcode *schedkwhennamed* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici un exemple de l'opcode `schedkwhennamed`. Il utilise le fichier `schedkwhennamed.csd` [examples/schedkwhennamed.csd].

### Exemple 479. Exemple de l'opcode `schedkwhennamed`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d
; For Non-realtime ouput leave only the line below:
; -o schedkwhennamed.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

    sr      = 48000
    ksmpps  = 16
    nchnls  = 2
    odbfs   = 1

; Example by Jonathan Murphy 2007

    gSinstr2 = "printer"

    instr 1

        ktrig    metro      1
    if (ktrig == 1) then
        ; Call instrument "printer" once per second
        schedkwhennamed ktrig, 0, 1, gSinstr2, 0, 1
    endif

    endin

    instr printer

        ktime    timeinsts
        printk2   ktime

    endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*event, event\_i, schedule, schedwhen, schedkwhen, scoreline, scoreline\_i*

## Crédits

Auteur : Rasmus Ekman  
EMS, Stockholm, Suède

Nouveau dans la version 4.23 de Csound

# schedule

`schedule` — Ajoute un nouvel évènement de partition.

## Description

Ajoute un nouvel évènement de partition.

## Syntaxe

```
schedule insnum, iwhen, idur [, ip4] [, ip5] [...]
```

```
schedule "insname", iwhen, idur [, ip4] [, ip5] [...]
```

## Initialisation

*insnum* -- numéro d'un instrument. Equivalent à p1 dans une *instruction i* de partition. *insnum* doit être un numéro supérieur au numéro de l'instrument appelant.

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*iwhen* -- date de début du nouvel évènement. Equivalent à p2 dans une *instruction i* de partition. *iwhen* ne doit pas être négatif. Si *iwhen* vaut zéro, *insnum* doit être supérieur ou égal au p1 de l'instrument courant.

*idur* -- durée de l'évènement. Equivalent à p3 dans une *instruction i* de partition.

*ip4*, *ip5*, ... -- Equivalent à p4, p5, etc., dans une *instruction i* de partition.

## Exécution

*trigger* -- valeur de déclenchement pour le nouvel évènement.

`schedule` ajoute un nouvel évènement de partition. Les arguments, options incluses, sont les mêmes que dans une partition. Le temps *iwhen* (p2) est mesuré à partir de l'instant de cet évènement.

Si la durée est nulle ou négative, le nouvel évènement est de type MIDI, et il hérite le sous-évènement de relachement (release) de l'instruction `schedule`.



### Note

Noter que l'opcode `schedule` ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode `scoreline` ou `scoreline_i`.

## Exemples

Voici un exemple de l'opcode `schedule`. Il utilise le fichier `schedule.csd` [examples/schedule.csd].

### Exemple 480. Exemple de l'opcode `schedule`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o schedule.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
; Play Instrument #2 at the same time.
schedule 2, 0, p3

; Play a high note.
a1 oscils 10000, 880, 1
out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
; Play a low note.
a1 oscils 10000, 220, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for half a second.
i 1 0 0.5
; Play Instrument #1 for half a second.
i 1 1 0.5
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*event, event\_i, schedwhen, schedkwhen, schedkwhennamed, scoreline, scoreline\_i*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Novembre 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.491 de Csound

Basé sur un travail de Gabriel Maldonado

Merci à David Gladstein, pour avoir clarifié le paramètre *iwhen*.

# schedwhen

schedwhen — Ajoute un nouvel évènement de partition.

## Description

Ajoute un nouvel évènement de partition.

## Syntaxe

```
schedwhen ktrigger, kinsnum, kwhen, kdur [, ip4] [, ip5] [...]
```

```
schedwhen ktrigger, "insname", kwhen, kdur [, ip4] [, ip5] [...]
```

## Initialisation

*ip4, ip5, ...* -- Equivalent à *p4, p5, etc.*, dans une *instruction i* de partition.

## Exécution

*kinsnum* -- numéro d'un instrument. Equivalent à *p1* dans une *instruction i* de partition.

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*ktrigger* -- valeur de déclenchement pour le nouvel évènement.

*kwhen* -- date de début du nouvel évènement. Equivalent à *p2* dans une *instruction i* de partition.

*kdur* -- durée de l'évènement. Equivalent à *p3* dans une *instruction i* de partition.

*schedwhen* ajoute un nouvel évènement de partition. L'évènement n'est programmé que lorsque la valeur de taux-*k* *ktrigger* prend une valeur non nulle. Les arguments, options incluses, sont les mêmes que dans une partition. Le temps *kwhen* (*p2*) est mesuré à partir de l'instant de cet évènement.

Si la durée est nulle ou négative, le nouvel évènement est de type MIDI, et il hérite le sous-évènement de relachement (release) de l'instruction *schedwhen*.



### Note

Noter que l'opcode *schedwhen* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici une exemple de l'opcode *schedwhen*. Il utilise le fichier *schedwhen.csd* [examples/schedwhen.csd].

**Exemple 481. Exemple de l'opcode schedwhen.**



Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o schedwhen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
; Use the fourth p-field as the trigger.
ktrigger = p4
kinsnum = 2
kwhen = 0
kdur = p3

; Play Instrument #2 at the same time, if the trigger is set.
schedwhen ktrigger, kinsnum, kwhen, kdur

; Play a high note.
a1 oscils 10000, 880, 1
out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
; Play a low note.
a1 oscils 10000, 220, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = trigger for Instrument #2 (when p4 > 0).
; Play Instrument #1 for half a second, trigger Instrument #2.
i 1 0 0.5 1
; Play Instrument #1 for half a second, no trigger.
i 1 1 0.5 0
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*event, event\_i, schedule, schedkwhen, schedkwhennamed, scoreline, scoreline\_i*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK

Novembre 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.491 de Csound

Basé sur un travail de Gabriel Maldonado

# seed

seed — Fixe la valeur globale de la graine.

## Description

Fixe la valeur globale de la graine pour tous les *générateurs de bruit de classe x*, ainsi que pour d'autres opcodes qui utilisent un appel de random, tels que *grain*.



### Noter que

*rand*, *randh*, *randi*, *rnd(x)* et *birnd(x)* ne sont pas affectés par *seed*.

## Syntaxe

```
seed ival
```

## Exécution

Avec l'utilisation de *seed* on obtiendra des résultats prévisibles d'un orchestre utilisant des générateurs de nombres aléatoires, lors de plusieurs exécutions.

Lors de la spécification d'une valeur de graine, *ival* doit être un entier compris entre 0 et  $2^{32}$ . Si *ival* = 0, la valeur de *ival* sera dérivée de l'horloge du système.

# sekere

sekere — Modèle semi-physique d'un son de chekeré.

## Description

*sekere* est un modèle semi-physique d'un son de chekeré. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares sekere iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 64.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,998 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,999 ce qui signifie que la valeur par défaut de *idamp* est 0,5. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode sekere. Il utilise le fichier *sekere.csd* [examples/sekere.csd].

### Exemple 482. Exemple de l'opcode sekere.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;;RT audio I/O
```

```

; For Non-realtime ouput leave only the line below:
; -o sekere.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra -----

    sr =          44100
    kr =          4410
    ksmps =        10
    nchnls =        1

instr 01                                ;an example of a sekere
a1      sekere p4, 0.01
        out a1
        endin

</CsInstruments>
<CsScore>

;score -----

i1 0 1 26000
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*cabasa, crunch, sandpaper, stix*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)  
 Adapté par John ffitch  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# semitone

semitone — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de demi-tons.

## Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de demi-tons.

## Syntaxe

`semitone(x)`

Cette fonction travaille aux taux-i, -k et -a.

## Initialisation

*x* -- une valeur exprimée en demi-tons.

## Exécution

La valeur retournée par la fonction *semitone* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre de demi-tons spécifié.

## Exemples

Voici un exemple de l'opcode demi-ton. Il utilise le fichier *semitone.csd* [examples/semitone.csd].

### Exemple 483. Exemple de l'opcode demi-ton.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o semitone.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; The root note is A above middle-C (440 Hz)
iroot = 440

; Raise the root note by three semitones to C.
isemitone = 3

; Calculate the new note.
```

```
ifactor = semitone(isemitone)
inew = iroot * ifactor

; Print out all of the values.
print iroot
print ifactor
print inew
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1: iroot = 440.000
instr 1: ifactor = 1.189
instr 1: inew = 523.229
```

## Voir Aussi

*cent, db, octave*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.16

## sense

sense — Same as the sensekey opcode.

## Description

Same as the *sensekey* opcode.



# sensekey

sensekey — Returns the ASCII code of a key that has been pressed.

## Description

Returns the ASCII code of a key that has been pressed, or -1 if no key has been pressed.

## Syntax

```
kres[, kkeydown] sensekey
```

## Performance

*kres* - returns the ASCII value of a key which is pressed or released.

*kkeydown* - returns 1 if the key was pressed, 0 if it was released or if there is no key event.

*kres* can be used to read keyboard events from stdin and returns the ASCII value of any key that is pressed or released, or it returns -1 when there is no keyboard activity. The value of *kkeydown* is 1 when a key was pressed, or 0 otherwise. This behavior is emulated by default, so a key release is generated immediately after every key press. To have full functionality, FLTK can be used to capture keyboard events. *FLpanel* can be used to capture keyboard events and send them to the sensekey opcode, by adding an additional optional argument. See *FLpanel* for more information.



### Note

This opcode can also be written as *sense*.

## Examples

Here is an example of the sensekey opcode. It uses the file *sensekey.csd* [examples/sensekey.csd].

### Exemple 484. Example of the sensekey opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o sensekey.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```

; Instrument #1.
instr 1
  k1 sensekey
  printk2 k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for thirty seconds.
i 1 0 30
e

</CsScore>
</CsoundSynthesizer>

```

Here is what the output should look like when the "q" button is pressed...

```
q i1 113.00000
```

Here is an example of the sensekey opcode in conjunction with *FLpanel*. It uses the file *FLpanel-sensekey.csd* [examples/FLpanel-sensekey.csd].

### Exemple 485. Example of the sensekey opcode using keyboard capture from an FLpanel.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in No messages
-odac -iadc -d ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLpanel-sensekey.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; Example by Johnathan Murphy

sr = 44100
ksmps = 128
nchnls = 2

; ikbdcapture flag set to 1
ikey init 1

gkasc, giasc FLbutBank 2, 16, 8, 700, 300, 20, 20, -1
FLpanelEnd
FLrun

instr 1
  kkey sensekey
  kprint changed kkey
  FLsetVal kprint, kkey, giasc

endin

</CsInstruments>
<CsScore>
i1 0 60
e
</CsScore>
</CsoundSynthesizer>

```

The lit button in the FLpanel window shows the last key pressed.

Here is a more complex example of the sensekey opcode in conjunction with *FLpanel*. It uses the file *FLpanel-sensekey2.csd* [examples/FLpanel-sensekey.csd].

### Exemple 486. Example of the sensekey opcode using keyboard capture from an FLpanel.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      ; -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLpanel-sensekey2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 48000
ksmps = 32
nchnls = 1
; Example by Istvan Varga
; if the FLTK opcodes are commented out, sensekey will read keyboard
; events from stdin
    FLpanel "", 150, 50, 100, 100, 0, 1
    FLlabel 18, 10, 1, 0, 0, 0
    FLgroup "Keyboard Input", 150, 50, 0, 0, 0
    FLgroupEnd
    FLpanelEnd

    FLrun

    instr 1

ktrig1 init 1
ktrig2 init 1
nxtKey1:
k1, k2 sensekey
    if (k1 != -1 || k2 != 0) then
        printf "Key code = %02X, state = %d\n", ktrig1, k1, k2
    ktrig1 = 3 - ktrig1
    kgoto nxtKey1
    endif
nxtKey2:
k3 sensekey
    if (k3 != -1) then
        printf "Character = '%c'\n", ktrig2, k3
    ktrig2 = 3 - ktrig2
    kgoto nxtKey2
    endif

    endin

</CsInstruments>
<CsScore>
i 1 0 3600
e
</CsScore>
</CsoundSynthesizer>
```

The console output will look something like:

```
new alloc for instr 1:
Key code = 65, state = 1
Character = 'e'
Key code = 65, state = 0
Key code = 72, state = 1
Character = 'r'
Key code = 72, state = 0
Key code = 61, state = 1
Character = 'a'
```

Key code = 61, state = 0

## See also

*FLpanel, FLkeyIn*

## Credits

Author: John ffitch  
University of Bath, Codemist. Ltd.  
Bath, UK  
October 2000

Examples written by Kevin Conder, Johnathan Murphy and Istvan Varga.

New in Csound version 4.09. Renamed in Csound version 4.10.

# seqtime

seqtime — Generates a trigger signal according to the values stored in a table.

## Description

Generates a trigger signal according to the values stored in a table.

## Syntax

```
ktrig_out seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times
```

## Performance

*ktrig\_out* -- output trigger signal

*ktime\_unit* -- unit of measure of time, related to seconds.

*kstart* -- start index of looped section

*kloop* -- end index of looped section

*kinitndx* -- initial index



### Note

Although *kinitndx* is listed as k-rate, it is in fact accessed only at init-time. So if you are using a k-rate argument, it must be assigned with *init*.

*kfn\_times* -- number of table containing a sequence of times

This opcode handles timed-sequences of groups of values stored into a table.

*seqtime* generates a trigger signal (a sequence of impulses, see also *trigger* opcode), according to the values stored in the *kfn\_times* table. This table should contain a series of delta-times (i.e. times between adjacent events). The time units stored into table are expressed in seconds, but can be rescaled by means of *ktime\_unit* argument. The table can be filled with *GEN02* or by means of an external text-file containing numbers, with *GEN23*.



### Note

Note that the *kloop* index marks the loop boundary and is NOT included in the looped elements. If you want to loop the first four elements, you would set *kstart* to 0 and *kloop* to 4.

It is possible to start the sequence from a value different than the first, by assigning to *kinitndx* an index different than zero (which corresponds to the first value of the table). Normally the sequence is looped, and the start and end of loop can be adjusted by modifying *kstart* and *kloop* arguments. User must be sure that values of these arguments (as well as *kinitndx*) correspond to valid table numbers, otherwise Csound will crash (because no range-checking is implemented).

It is possible to disable loop (one-shot mode) by assigning the same value both to *kstart* and *kloop* argu-

ments. In this case, the last read element will be the one corresponding to the value of such arguments. Table can be read backward by assigning a negative *kloop* value. It is possible to trigger two events almost at the same time (actually separated by a k-cycle) by giving a zero value to the corresponding delta-time. First element contained in the table should be zero, if the user intends to send a trigger impulse, it should come immediately after the orchestra instrument containing *seqtime* opcode.

## Examples

Here is an example of the *seqtime* opcode. It uses the file *seqtime.csd* [examples/seqtime.csd].

### Exemple 487. Example of the *seqtime* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o seqtime.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 64
nchnls = 1

; By Tim Mortimer and Andres Cabrera 2007

0dbfs = 1

gisine      ftgen      0, 0, 8192, 10,      1
;;; table defining an integer pitch set
gipset      ftgen      0, 0, 4, -2, 8.00, 8.04, 8.07, 8.10
;;;DELTA times for seqtime
gidelta     ftgen      0, 0, 4, -2, .5, 1, .25, 1.25

instr 1
kndx init 0
ktrigger init 0

ktime_unit init 1
kstart init p4
kloop init p5
kinitndx init 0
kfn_times init gidelta

ktrigger seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times

printk2 ktrigger

if (ktrigger > 0) then
  kpitch table kndx, gipset
  event "i", 2, 0, 1, kpitch
  kndx = kndx + 1
  kndx = kndx % kloop
endif
endin

instr 2
icps = cpspch (p4)
a1 buzz 1, icps, 7, gisine
aamp expseg 0.00003, .02, 1, p3-.02, 0.00003
a1 = a1 * aamp * 0.5
```

```
out a1
    endin

</CsInstruments>
<CsScore>
;      start      dur      kstart      kloop
i 1 0 7 0 4
i 1 8 10 0 3
i 1 19 10 4 4

</CsScore>
</CsoundSynthesizer>
```

## See Also

*GEN02*, *GEN23*, *trigseq seqtime2*

## Credits

Author: Gabriel Maldonado

November 2002. Added a note about the *kinitndx* parameter, thanks to Rasmus Ekman.

New in version 4.06

Example by: Tim Mortimer and Andres Cabrera 2007

# seqtime2

seqtime2 — Generates a trigger signal according to the values stored in a table.

## Description

Generates a trigger signal according to the values stored in a table.

## Syntax

```
ktrig_out seqtime2 ktrig_in, ktime_unit, kstart, kloop, kinitndx, kfn_times
```

## Performance

*ktrig\_out* -- output trigger signal

*ktime\_unit* -- unit of measure of time, related to seconds.

*ktime\_in* -- input trigger signal.

*kstart* -- start index of looped section

*kloop* -- end index of looped section

*kinitndx* -- initial index



### Note

Although *kinitndx* is listed as k-rate, it is in fact accessed only at init-time. So if you are using a k-rate argument, it must be assigned with *init*.

*kfn\_times* -- number of table containing a sequence of times

This opcode handles timed-sequences of groups of values stored into a table.

*seqtime2* generates a trigger signal (a sequence of impulses, see also *trigger* opcode), according to the values stored in the *kfn\_times* table. This table should contain a series of delta-times (i.e. times between adjacent events). The time units stored into table are expressed in seconds, but can be rescaled by means of *ktime\_unit* argument. The table can be filled with *GEN02* or by means of an external text-file containing numbers, with *GEN23*.

It is possible to start the sequence from a value different than the first, by assigning to *initndx* an index different than zero (which corresponds to the first value of the table). Normally the sequence is looped, and the start and end of loop can be adjusted by modifying *kstart* and *kloop* arguments. User must be sure that values of these arguments (as well as *initndx*) correspond to valid table numbers, otherwise Csound will crash (because no range-checking is implemented).

It is possible to disable loop (one-shot mode) by assigning the same value both to *kstart* and *kloop* arguments. In this case, the last read element will be the one corresponding to the value of such arguments. Table can be read backward by assigning a negative *kloop* value. It is possible to trigger two events almost at the same time (actually separated by a k-cycle) by giving a zero value to the corresponding delta-time. First element contained in the table should be zero, if the user intends to send a trigger impulse, it should come immediately after the orchestra instrument containing *seqtime2* opcode.



*seqtime2* is similar to *seqtime*, the difference is that when *ktrig\_in* contains a non-zero value, current index is reset to *kinitndx* value. *kinitndx* can be varied at performance time.

## See Also

*GEN02*, *GEN23*, *seqtime*, *trigseq*, *timedseq*

## Credits

Author: Gabriel Maldonado

# setctrl

setctrl — Configurable slider controls for realtime user input.

## Description

Configurable slider controls for realtime user input. Requires Winsound or TCL/TK. *setctrl* sets a slider to a specific value, or sets a minimum or maximum range.

## Syntax

```
setctrl inum, ival, itype
```

## Initialization

*inum* -- number of the slider to set

*ival* -- value to be sent to the slider

*itype* -- type of value sent to the slider as follows:

- 1 -- set the current value. Initial value is 0.
- 2 -- set the minimum value. Default is 0.
- 3 -- set the maximum value. Default is 127.
- 4 -- set the label. (New in Csound version 4.09)

## Performance

Calling *setctrl* will create a new slider on the screen. There is no theoretical limit to the number of sliders. Windows and TCL/TK use only integers for slider values, so the values may need rescaling. GUIs usually pass values at a fairly slow rate, so it may be advisable to pass the output of control through *port*.

## Examples

Here is an example of the setctrl opcode. It uses the file *setctrl.csd* [examples/setctrl.csd].

### Exemple 488. Example of the setctrl opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```

; -o setctrl.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Display the label "Volume" on Slider #1.
setctrl 1, "Volume", 4
; Set Slider #1's initial value to 20.
setctrl 1, 20, 1

; Capture and display the values for Slider #1.
k1 control 1
printk2 k1

; Play a simple oscillator.
; Use the values from Slider #1 for amplitude.
kamp = k1 * 128
a1 oscil kamp, 440, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for thirty seconds.
i 1 0 30
e

</CsScore>
</CsoundSynthesizer>

```

Its output should include lines like this:

```

i1      38.00000
i1      40.00000
i1      43.00000

```

## See Also

*control*

## Credits

Author: John ffitch  
 University of Bath, Codemist. Ltd.  
 Bath, UK  
 May 2000

Example written by Kevin Conder.

New in Csound version 4.06

# setksmps

setksmps — Fixe la valeur locale de ksmmps dans un bloc d'opcode défini par l'utilisateur.

## Description

Fixe la valeur locale de *ksmps* dans un bloc d'opcode défini par l'utilisateur.

## Syntaxe

```
setksmps iksmps
```

## Initialisation

*iksmps* -- fixe la valeur locale de *ksmps*.

Si *iksmps* vaut zéro, le *ksmps* de l'instrument ou de l'opcode appelant est utilisé (c'est le comportement par défaut).



### Note

Le *ksmps* local est implémenté en divisant une période de contrôle en sous-périodes-k plus petites et en modifiant temporairement les variables globales internes de Csound. Ceci nécessite également de convertir le taux des entrées de taux-k et des arguments de sortie (les variables d'entrée reçoivent la même valeur durant toutes les sous-périodes-k, tandis que les sorties ne sont écrites que dans la dernière). Cela signifie aussi que l'on ne peut pas utiliser un *ksmps* local supérieur au *ksmps* global.



### Avertissement au sujet du *ksmps* local

Lorsque le *ksmps* local est différent de celui de l'orchestre (défini dans l'en-tête de l'orchestre), il ne faut pas utiliser d'opérations globales de taux-a dans le bloc d'opcode défini par l'utilisateur.

Ça comprend :

- tout accès aux variables « ga »
- les opcodes zak de taux-a (*zar*, *zaw*, etc.)
- *tablera* et *tablewa* (en fait, ces deux opcodes peuvent fonctionner, mais il faut prendre des précautions)
- La famille d'opcodes *in* et *out* (ceux-ci lisent et écrivent dans des tampons globaux de taux-a)

En général, le *ksmps* local doit être utilisé avec précaution car c'est un dispositif expérimental. Bien qu'il fonctionne dans la plupart des cas.

On peut utiliser l'instruction *setksmps* pour fixer la valeur locale de *ksmps* dans un bloc d'opcode défini

par l'utilisateur. Il a un paramètre de taux-*i* définissant la nouvelle valeur de *ksmps* (qui reste inchangée si l'on utilise zéro). *setksmps* doit être utilisé avant tout autre opcode (mais on peut le mettre après *xin*), sinon il y aura des résultats imprévisibles.

## Exécution

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

On peut alors utiliser le nouvel opcode avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN] name [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

## Exemples

Voir l'exemple de l'opcode *opcode*.

## Voir Aussi

*endop*, *opcode*, *xin*, *xout*

## Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code par Matt J. Ingalls

Nouveau dans la version 4.22

# setscorepos

setscorepos — Sets the playback position of the current score performance to a given position.

## Description

Sets the playback position of the current score performance to a given position.

## Syntax

```
setscorepos ipos
```

## Initialization

*ipos* -- playback position in seconds.

## Examples

Here is an example of the setscorepos opcode.

### Exemple 489. Example of the setscorepos opcode.

```
instr 1
  setscorepos 10
endin
```

## See Also

*setscorepos*,

## Credits

Author: Victor Lazzarini  
2008

New in Csound version 5.09

# sfilist

sfilist — Imprime une liste de tous les instruments d'un fichier SoundFont2 (SF2) préalablement chargé.

## Description

Imprime une liste de tous les instruments d'un fichier SoundFont2 (SF2) de sons échantillonnés préalablement chargé. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
sfilist ifilhandle
```

## Initialisation

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

## Exécution

*sfilist* imprime sur la console une liste de tous les instruments d'un fichier SoundFont2 (SF2) préalablement chargé.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfinstr

sfinstr — Joue un instrument échantillonné SoundFont2 (SF2), produisant un son stéréo.

## Description

Joue un instrument échantillonné SoundFont2 (SF2), produisant un son stéréo. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ar1, ar2 sfinstr ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \  
[, iflag] [, ioffset]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*instrnum* -- numéro d'un instrument d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Le paramètre *ioffset* permet de commencer la lecture depuis un autre échantillon que le premier. L'utilisateur doit s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

*sfinstr* joue un instrument SF2 plutôt qu'un preset (un instrument SF2 est la base d'une couche de preset).



*instrnum* indique le numéro de l'instrument, et l'utilisateur doit s'assurer que le numéro spécifié est celui d'un instrument existant d'une banque soundfont déterminée. Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a, mais les deux arguments doivent travailler au même taux.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist*, *sfinstrm*, *sfloat*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfinstr3

sfinstr3 — Joue un instrument échantillonné SoundFont2 (SF2), produisant un son stéréo avec interpolation cubique.

## Description

Joue un instrument échantillonné SoundFont2 (SF2), produisant un son stéréo avec interpolation cubique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ar1, ar2 sfinstr3 ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
[, iflag] [, ioffset]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*instrnum* -- numéro d'un instrument d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Le paramètre *ioffset* permet de commencer la lecture depuis un autre échantillon que le premier. L'utilisateur doit s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

*sfinstr3* est une version avec interpolation cubique de *sfinstr*. La différence de qualité sonore est notable, particulièrement avec les échantillons transposés dans le grave. Pour les échantillons transposés dans l'aigu, la différence est moins appréciable et je suggère d'utiliser les versions avec interpolation linéaire, car elles sont plus rapides.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist, sfinstr3m, sfinstrm, sfinstr, sfload, sfpassign, sfplay3, sfplay3m, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfinstr3m

**sfinstr3m** — Joue un instrument échantillonné SoundFont2 (SF2), produisant un son mono avec interpolation cubique.

## Description

Joue un instrument échantillonné SoundFont2 (SF2), produisant un son mono avec interpolation cubique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ares sfinstr3m ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \  
    [, iflag] [, ioffset]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*instrnum* -- numéro d'un instrument d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Le paramètre *ioffset* permet de commencer la lecture depuis un autre échantillon que le premier. L'utilisateur doit s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

*sfinstr3m* est une version avec interpolation cubique de *sfinstrm*. La différence de qualité sonore est notable, particulièrement avec les échantillons transposés dans le grave. Pour les échantillons transposés dans l'aigu, la différence est moins appréciable et je suggère d'utiliser les versions avec interpolation linéaire, car elles sont plus rapides.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist*, *sfinstr3*, *sfinstr*, *sfinstrm*, *sload*, *sfpassign*, *sfplay3*, *sfplay3m*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfinstrm

sfinstrm — Joue un instrument échantillonné SoundFont2 (SF2), produisant un son mono.

## Description

Joue un instrument échantillonné SoundFont2 (SF2), produisant un son mono. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ares sfinstrm ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \  
    [, iflag] [, ioffset]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*instrnum* -- numéro d'un instrument d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Le paramètre *ioffset* permet de commencer la lecture depuis un autre échantillon que le premier. L'utilisateur doit s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

*sfinstrm* est une version mono de *sfinstr*. C'est l'opcode le plus rapide de la famille SF2.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist, sfinstr, sfload, sfpassign, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfload

sfload — Charge en mémoire un fichier d'échantillons SoundFont2 (SF2) en entier.

## Description

Charge en mémoire un fichier d'échantillons SoundFont2 (SF2) en entier. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

*sfload* doit être placé dans la section d'en-tête d'un orchestre de Csound.

## Syntaxe

```
ir sfload "filename"
```

## Initialisation

*ir* -- valeur de retour à utiliser par les autres opcodes SF2. Pour *sfload*, *ir* est le *ifilhandle*.

« *filename* » -- nom du fichier SF2, avec son chemin complet. C'est une chaîne de caractères entre guillemets avec le « / » comme séparateur de répertoires (ceci s'applique également à DOS et à Windows, où l'utilisation d'un antislash générera une erreur), ou bien un entier qui a été lié à une chaîne de caractères par *strset*.

## Exécution

*sfload* charge en mémoire un fichier SF2 en entier. Il retourne un identificateur de fichier à utiliser par les autres opcodes. On peut placer plusieurs instances de *sfload* dans la section d'en-tête d'un orchestre, ce qui permet d'utiliser plusieurs fichiers SF2 dans un seul orchestre.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

Il faut noter qu'avant la version 5.12 on pouvait charger au maximum dix soundfonts. Cette restriction est maintenant levée.

## Voir Aussi

*sfilist*, *sfinstr*, *sfinstrm*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound



# sflooper

**sflooper** — Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo, avec une boucle en fondu-enchaîné à durée variable, définie par l'utilisateur.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo, comme *sfplay*. Mais à l'inverse de ce dernier, il ignore les points de boucle fixés dans le fichier SF2 et les remplace par une boucle en fondu-enchaîné définie par l'utilisateur. C'est un mélange de *sfplay* et de *flooper2*.

## Syntaxe

```
ar1, ar2 sflooper ivel, inotenum, kamp, kpitch, ipreindex, kloopstart, kloopend, kcrossfade, ifn \
[, istart, imode, ifenv, iskip]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice de preset.

*istart* -- début de la lecture en secondes.

*imode* -- modes de boucle : 0 à l'endroit, 1 à l'envers, 2 à l'envers et à l'endroit [0 par défaut].

*ifenv* -- s'il est différent de zéro, numéro de la table de l'enveloppe de fondu-enchaîné. La valeur par défaut de 0 définit un fondu-enchaîné linéaire.

*iskip* -- s'il vaut 1, l'initialisation de l'opcode est ignorée, pour les notes liées, l'exécution continuant depuis la position dans la boucle où la note précédente s'est terminée. Avec la valeur par défaut de 0, l'initialisation a lieu.

## Exécution

*kamp* -- contrôle de l'amplitude

*kpitch* -- contrôle de la hauteur (rapport de transposition) ; les valeurs négatives sont interdites.

*kloopstart* -- début de la boucle (en secondes). Noter que bien qu'étant de taux-k, les paramètres de boucle comme celui-ci ne sont mis à jour qu'une fois par itération de la boucle. Si le début de la boucle est fixé au-delà de la fin des échantillons, il n'y aura pas de boucle.

*kloopend* -- fin de la boucle (en secondes), mis à jour une seule fois par itération de la boucle.

*kcrossfade* -- longueur du fondu enchaîné (en secondes), mis à jour une seule fois par itération de la boucle et limité par la longueur de la boucle.

*sflooper* joue un preset, générant un son stéréo.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données

échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist, sfinstr, sfinstrm, sfload, sfpassign, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Victor Lazzarini  
Août 2007

Nouveau dans la version 5.07 de Csound

# sfpassign

**sfpassign** — Associe tous les presets d'un fichier d'échantillons SoundFont2 (SF2) à une suite croissante d'indices numériques.

## Description

Associe tous les presets d'un fichier d'échantillons SoundFont2 (SF2) préalablement chargé en mémoire à une suite croissante d'indices numériques. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

*sfpassign* doit être placé dans le section d'en-tête d'un orchestre de Csound.

## Syntaxe

```
sfpassign istartindex, ifilhandle[, imsgs]
```

## Initialisation

*istartindex* -- indice de départ de la séquence à associer à l'ensemble des presets.

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*imsgs* -- s'il est différent de zéro, les messages sont supprimés.

## Exécution

*sfpassign* associe tous les presets d'un fichier d'échantillons SoundFont2 (SF2) préalablement chargé en mémoire à une suite croissante d'indices numériques, à utiliser ensuite avec les opcodes *sfplay* et *sfplaym*. *istartindex* indique la valeur du premier indice. On peut placer plusieurs instances de *sfpassign* dans la section d'en-tête d'un orchestre, chacune associant une séquence d'indices aux presets d'un fichier SF2 différent. L'utilisateur doit veiller à ce que les valeurs d'indice des preset de fichiers SF2 différents soient disjointes.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist*, *sfinstr*, *sfinstrm*, *sfload*, *sfplay*, *sfplaym*, *sfplist*, *sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfplay

sfplay — Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ar1, ar2 sfplay ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice du preset.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

*ienv* (facultatif) -- active et détermine l'enveloppe d'amplitude. 0 = pas d'enveloppe, 1 = attaque et chute linéaires, 2 = attaque linéaire, chute exponentielle (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a. Les deux arguments doivent utiliser des variables de même taux, sinon *sfplay* ne fonctionnera pas correctement. *ipreindex* doit contenir un numéro associé préalablement à un preset, ou Csound se plantera.

Le paramètre *ioffset* permet de démarrer le son depuis un autre échantillon que le premier. Il faut s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

Le paramètre *ienv* active et détermine l'enveloppe d'amplitude utilisée. Sa valeur par défaut est 0, soit pas d'enveloppe. Si *ienv* vaut 1, les portions de l'attaque et de la chute sont linéaires. S'il vaut 2, l'attaque est linéaire et la chute est exponentielle. La portion de relâchement de l'enveloppe n'a pas encore été implémentée.

*sfplay* joue un preset, générant un son stéréo. *ivel* n'affecte pas directement l'amplitude de la sortie, mais indique à *sfplay* quels échantillons choisir dans les presets à sons échantillonnés multiples, séparés par la vélocité.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist, sfinstr, sfinstrm, sfload, sfpassign, sfplay3, sfplaym, sfplay3m, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

Nouveau paramètre facultatif *ienv* dans la version 5.09

# sfplay3

`sfplay3` — Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo avec interpolation cubique.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo avec interpolation cubique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ar1, ar2 sfplay3 ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice du preset.

*iflag* -- (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

*ienv* (facultatif) -- active et détermine l'enveloppe d'amplitude. 0 = pas d'enveloppe, 1 = attaque et chute linéaires, 2 = attaque linéaire, chute exponentielle (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a. Les deux arguments doivent utiliser des variables de même taux, sinon `sfplay3` ne fonctionnera pas correctement. *ipreindex* doit contenir un numéro associé préalablement à un preset, ou Csound se plantera.

Le paramètre *ioffset* permet de démarrer le son depuis un autre échantillon que le premier. Il faut

s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

Le paramètre *ienv* active et détermine l'enveloppe d'amplitude utilisée. Sa valeur par défaut est 0, soit pas d'enveloppe. Si *ienv* vaut 1, les portions de l'attaque et de la chute sont linéaires. S'il vaut 2, l'attaque est linéaire et la chute est exponentielle. La portion de relâchement de l'enveloppe n'a pas encore été implémentée.

*sfplay3* joue un preset, générant un son stéréo avec interpolation cubique. *ivel* n'affecte pas directement l'amplitude de la sortie, mais indique à *sfplay* quels échantillons choisir dans les presets à sons échantillonnés multiples, séparés par la vitesse.

*sfplay3* est une version de *sfplay* avec interpolation cubique. La différence de qualité sonore est notable, particulièrement avec les échantillons transposés dans le grave. Pour les échantillons transposés dans l'aigu, la différence est moins appréciable et je suggère d'utiliser les versions avec interpolation linéaire, car elles sont plus rapides.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist*, *sfinstr3*, *sfinstr3m*, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay3m*, *sfplaym*, *sfplay*, *sfplist*, *sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

Nouveau paramètre facultatif *ienv* dans la version 5.09



# sfplay3m

**sfplay3m** — Joue un preset d'échantillons SoundFont2 (SF2), générant un son mono avec interpolation cubique.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son mono avec interpolation cubique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ares sfplay3m ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice du preset.

*iflag* (optional) -- (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

*ienv* (facultatif) -- active et détermine l'enveloppe d'amplitude. 0 = pas d'enveloppe, 1 = attaque et chute linéaires, 2 = attaque linéaire, chute exponentielle (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a. Les deux arguments doivent utiliser des variables de même taux, sinon *sfplay3m* ne fonctionnera pas correctement. *ipreindex* doit contenir un numéro associé préalablement à un preset, ou Csound se plantera.

Le paramètre *ioffset* permet de démarrer le son depuis un autre échantillon que le premier. Il faut

s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

Le paramètre *ienv* active et détermine l'enveloppe d'amplitude utilisée. Sa valeur par défaut est 0, soit pas d'enveloppe. Si *ienv* vaut 1, les portions de l'attaque et de la chute sont linéaires. S'il vaut 2, l'attaque est linéaire et la chute est exponentielle. La portion de relâchement de l'enveloppe n'a pas encore été implémentée.

*sfplay3m* est une version mono de *sfplay3*. Il faut l'utiliser avec un preset mono, ou avec des presets stéréo dans lesquels la sortie stéréo n'est pas requise. Il est plus rapide que *sfplay3*.

*sfplay3m* est aussi une version avec interpolation cubique de *sfplaym*. La différence de qualité sonore est notable, particulièrement avec les échantillons transposés dans le grave. Pour les échantillons transposés dans l'aigu, la différence est moins appréciable et je suggère d'utiliser les versions avec interpolation linéaire, car elles sont plus rapides.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist*, *sfinstr3*, *sfinstr3m*, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay3*, *sfplaym*, *sfplay*, *sfplist*, *sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

Nouveau paramètre facultatif *ienv* dans la version 5.09

# sfplaym

sfplaym — Joue un preset d'échantillons SoundFont2 (SF2), générant un son mono.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son mono. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ares sfplaym ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice du preset.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

*ienv* (facultatif) -- active et détermine l'enveloppe d'amplitude. 0 = pas d'enveloppe, 1 = attaque et chute linéaires, 2 = attaque linéaire, chute exponentielle (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a. Les deux arguments doivent utiliser des variables de même taux, sinon *sfplay* ne fonctionnera pas correctement. *ipreindex* doit contenir un numéro associé préalablement à un preset, ou Csound se plantera.

Le paramètre *ioffset* permet de démarrer le son depuis un autre échantillon que le premier. Il faut s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

Le paramètre *ienv* active et détermine l'enveloppe d'amplitude utilisée. Sa valeur par défaut est 0, soit pas d'enveloppe. Si *ienv* vaut 1, les portions de l'attaque et de la chute sont linéaires. S'il vaut 2, l'attaque est linéaire et la chute est exponentielle. La portion de relâchement de l'enveloppe n'a pas encore été implémentée.

*sfplaym* est une version mono de *sfplay*. Il faut l'utiliser avec un preset mono, ou avec des presets stéréo dans lesquels la sortie stéréo n'est pas requise. Il est plus rapide que *sfplay*.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist, sfinstr, sfinstrm, sfload, sfpassign, sfplay, sfplay3, sfplay3m, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

Nouveau paramètre facultatif *ienv* dans la version 5.09

# sfplist

sfplist — Imprime une liste de tous les presets d'un fichier d'échantillons SoundFont2 (SF2).

## Description

Imprime une liste de tous les presets d'un fichier d'échantillons SoundFont2 (SF2) préalablement chargé en mémoire. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
sfplist ifilhandle
```

## Initialisation

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

## Exécution

*sfplist* imprime sur la console une liste de tous les presets d'un fichier (SF2) préalablement chargé en mémoire.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sflist, sfinstr, sfinstrm, sfload, sfpassign, sfplay, sfplaym, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfpreset

sfpreset — Associe un preset d'un fichier d'échantillons SoundFont2 (SF2) à un indice numérique.

## Description

Associe un preset d'un fichier d'échantillons SoundFont2 (SF2) préalablement chargé en mémoire à un indice numérique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

*sfpreset* doit être placé dans la section d'en-tête d'un orchestre de Csound.

## Syntaxe

```
ir sfpreset iprog, ibank, ifilhandle, ipreindex
```

## Initialisation

*ir* -- valeur de retour à utiliser par les autres opcodes SF2. Pour *sfpreset*, *ir* est le *ipreindex*.

*iprog* -- numéro de programme d'une banque de presets dans un fichier SF2.

*ibank* -- numéro d'une banque spécifique d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*ipreindex* -- indice du preset.

## Exécution

*sfpreset* associe un preset d'un fichier SF2 préalablement chargé en mémoire à un indice numérique, à utiliser ensuite avec les opcodes *sfplay* et *sfplaym*. L'utilisateur doit connaître à l'avance les numéros de banque du preset afin de remplir les arguments correspondants. On peut placer plusieurs instances de *sfpreset* dans la section d'en-tête d'un orchestre, chacune associant un preset différent appartenant au même (ou à différents) fichiers SF2 à différents indices numériques.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir Aussi

*sfilist*, *sfinstr*, *sfinstrm*, *sfload*, *sfpassign*, *sfplay*, *sfplaym*, *sfplist*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Mai 2000

Nouveau dans la version 4.07 de Csound

# shaker

shaker — Produit un son comme si l'on secouait des maracas ou un instrument similaire de type calebasse.

## Description

La sortie audio produit un son comme si l'on secouait des maracas ou un instrument similaire de type calebasse. La méthode est inspirée d'un modèle physique développé d'après Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares shaker kamp, kfreq, kbeans, kdamp, ktimes [, idecay]
```

## Initialisation

*idecay* -- S'il est présent, indique la durée d'amortissement du shaker à la fin de la note. La valeur par défaut est zéro.

## Exécution

Une note jouée sur un instrument de type maracas, avec les arguments suivants.

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kbeans* -- Le nombre de graines dans la calebasse. Une valeur de 8 est convenable.

*kdamp* -- La valeur d'amortissement du shaker. Des valeurs comprises entre 0,98 et 1 conviennent, avec une valeur raisonnable par défaut de 0,99.

*ktimes* -- Nombre de secousses.



### Note

L'argument *knum* était redondant et a donc été supprimé dans la version 3.49.

## Exemples

Voici un exemple de l'opcode shaker. Il utilise le fichier *shaker.csd* [examples/shaker.csd].

### Exemple 490. Exemple de l'opcode shaker.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```



```
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o shaker.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1
instr 1
  al shaker 10000, 440, 8, 0.999, 100, 0
  out al
endin

</CsInstruments>
<CsScore>

i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

Exemple corrigé grâce à un message de Istvan Varga.

# sin

sin — Calcule une fonction sinus.

## Description

Retourne sinus de  $x$  ( $x$  en radians).

## Syntaxe

**sin**( $x$ ) (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode sin. Il utilise le fichier *sin.csd* [examples/sin.csd].

### Exemple 491. Exemple de l'opcode sin.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 25
  i1 = sin(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = -0.132
```

## Voir Aussi

*cos, cosh, cosinv, sinh, sininv, tan, tanh, taninv*

## Crédits

Exemple écrit par Kevin Conder.

# sinh

sinh — Calcule une fonction sinus hyperbolique.

## Description

Retourne sinus hyperbolique de  $x$  ( $x$  en radians).

## Syntaxe

**sinh**( $x$ ) (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode sinh. Il utilise le fichier *sinh.csd* [examples/sinh.csd].

### Exemple 492. Exemple de l'opcode sinh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sinh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 1
  i1 = sinh(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsSoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.175
```

## Voir Aussi

*cos, cosh, cosinv, sin, sininv, tan, tanh, taninv*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47

# sininv

sininv — Calcule une fonction arcsinus.

## Description

Retourne arcsinus de  $x$  ( $x$  en radians).

## Syntaxe

**sininv**( $x$ ) (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode sininv. Il utilise le fichier *sininv.csd* [exemples/sininv.csd].

### Exemple 493. Exemple de l'opcode sininv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o sininv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = sininv(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.524
```

## Voir Aussi

*cos, cosh, cosinv, sin, sinh, tan, tanh, taninv*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

# sinsyn

sinsyn — Streaming partial track additive synthesis with cubic phase interpolation

## Description

The sinsyn opcode takes an input containing a TRACKS pv streaming signal (as generated, for instance by the partials opcode). It sinsynthesises the signal using linear amplitude and cubic phase interpolation to drive a bank of interpolating oscillators with amplitude and pitch scaling controls. Sinsyn attempts to preserve the phase of the partials in the original signal and in so doing it does not allow for pitch or timescale modifications of the signal.

## Syntax

```
asig sinsyn fin, kscal, kmaxtracks, ifn
```

## Performance

*asig* -- output audio rate signal

*fin* -- input pv stream in TRACKS format

*kscal* -- amplitude scaling

*kmaxtracks* -- max number of tracks in sinsynthesis. Limiting this will cause a non-linear filtering effect, by discarding newer and higher-frequency tracks (tracks are ordered by start time and ascending frequency, respectively)

*ifn* -- function table containing one cycle of a sinusoid (sine or cosine)

## Examples

### Exemple 494. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
    aout sinsyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows partial tracking of an ifd-analysis signal and cubic-phase additive resynthesis.

## Credits

Author: Victor Lazzarini  
June 2005



New plugin in version 5

November 2004.

# sleighbells

sleighbells — Modèle semi-physique d'un son de cloche de traîneau.

## Description

*sleighbells* est un modèle semi-physique d'un son de cloche de traîneau. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares sleighbells kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \  
    [, ifreq1] [, ifreq2]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 32.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$\text{damping\_amount} = 0,9994 + (\text{idamp} * 0,002)$

La valeur par défaut de *damping\_amount* est 0,9994 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,03.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2500.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 5300.

*ifreq2* (facultatif) -- la deuxième fréquence de résonance. La valeur par défaut est 6500.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode *sleighbells*. Il utilise le fichier *sleighbells.csd* [examples/sleighbells.csd].

### Exemple 495. Exemple de l'opcode sleighbells.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sleighbells.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1: An example of sleighbells.
instr 1
  al sleighbells 20000, 0.01

  out al
endin

</CsInstruments>
<CsScore>

i 1 0.00 0.25
i 1 0.30 0.25
i 1 0.60 0.25
i 1 0.90 0.25
i 1 1.20 0.25
i 1 1.50 0.25
i 1 1.80 0.25
i 1 2.10 0.25
i 1 2.40 0.25
i 1 2.70 0.25
i 1 3.00 0.25
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*bamboo, dripwater, guiro, tambourine*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)  
 Adapté par John ffitch  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# slider16

slider16 — Creates a bank of 16 different MIDI control message numbers.

## Description

Creates a bank of 16 different MIDI control message numbers.

## Syntax

```
i1,...,i16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
            ictlnum16, imin16, imax16, init16, ifn16  
  
k1,...,k16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
            ictlnum16, imin16, imax16, init16, ifn16
```

## Initialization

*i1 ... i16* -- output values

*ichan* -- MIDI channel (1-16)

*ictlnum1 ... ictlnum16* -- MIDI control number (0-127)

*imin1 ... imin16* -- minimum values for each controller

*imax1 ... imax16* -- maximum values for each controller

*init1 ... init16* -- initial value for each controller

*ifn1 ... ifn16* -- function table for conversion for each controller

*icutoff1 ... icutoff16* -- low-pass filter cutoff frequency for each controller

## Performance

*k1 ... k16* -- output values

*slider16* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider16* allows a bank of 16 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *slider16*, there is not an initial value input argument, because the output is gotten directly from current status of internal controller array of Csound.

## See Also

*s16b14*, *s32b14*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# slider16f

slider16f — Creates a bank of 16 different MIDI control message numbers, filtered before output.

## Description

Creates a bank of 16 different MIDI control message numbers, filtered before output.

## Syntax

```
k1,...,k16 slider16f ichan, ictlnum1, imin1, imax1, init1, ifn1, \  
            icutoff1,..., ictlnum16, imin16, imax16, init16, ifn16, icutoff16
```

## Initialization

*ichan* -- MIDI channel (1-16)

*ictlnum1* ... *ictlnum16* -- MIDI control number (0-127)

*imin1* ... *imin16* -- minimum values for each controller

*imax1* ... *imax16* -- maximum values for each controller

*init1* ... *init16* -- initial value for each controller

*ifn1* ... *ifn16* -- function table for conversion for each controller

*icutoff1* ... *icutoff16* -- low-pass filter cutoff frequency for each controller

## Performance

*k1* ... *k16* -- output values

*slider16f* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider16f* allows a bank of 16 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.



## Avertissement

*slider16f* does not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

## See Also

*s16b14*, *s32b14*, *slider16*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# slider32

slider32 — Creates a bank of 32 different MIDI control message numbers.

## Description

Creates a bank of 32 different MIDI control message numbers.

## Syntax

```
i1,...,i32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
            ictlnum32, imin32, imax32, init32, ifn32  
  
k1,...,k32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
            ictlnum32, imin32, imax32, init32, ifn32
```

## Initialization

*i1 ... i32* -- output values

*ichan* -- MIDI channel (1-16)

*ictlnum1 ... ictlnum32* -- MIDI control number (0-127)

*imin1 ... imin32* -- minimum values for each controller

*imax1 ... imax32* -- maximum values for each controller

*init1 ... init32* -- initial value for each controller

*ifn1 ... ifn32* -- function table for conversion for each controller

*icutoff1 ... icutoff32* -- low-pass filter cutoff frequency for each controller

## Performance

*k1 ... k32* -- output values

*slider32* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider32* allows a bank of 32 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.



In the i-rate version of *slider32*, there is not an initial value input argument, because the output is gotten directly from current status of internal controller array of Csound.

## See Also

*s16b14*, *s32b14*, *slider16*, *slider16f*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# slider32f

slider32f — Creates a bank of 32 different MIDI control message numbers, filtered before output.

## Description

Creates a bank of 32 different MIDI control message numbers, filtered before output.

## Syntax

```
k1,...,k32 slider32f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \  
..., ictlnum32, imin32, imax32, init32, ifn32, icutoff32
```

## Initialization

*ichan* -- MIDI channel (1-16)

*ictlnum1* ... *ictlnum32* -- MIDI control number (0-127)

*imin1* ... *imin32* -- minimum values for each controller

*imax1* ... *imax32* -- maximum values for each controller

*init1* ... *init32* -- initial value for each controller

*ifn1* ... *ifn32* -- function table for conversion for each controller

*icutoff1* ... *icutoff32* -- low-pass filter cutoff frequency for each controller

## Performance

*k1* ... *k32* -- output values

*slider32f* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider32f* allows a bank of 32 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.



## Avertissement

*slider32f* opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

## See Also

*s16b14, s32b14, slider16, slider16f, slider32, slider64, slider64f, slider8, slider8f*

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# slider64

slider64 — Creates a bank of 64 different MIDI control message numbers.

## Description

Creates a bank of 64 different MIDI control message numbers.

## Syntax

```
i1,...,i64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum64, imin64, imax64, init64, ifn64  
  
k1,...,k64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum64, imin64, imax64, init64, ifn64
```

## Initialization

*i1 ... i64* -- output values

*ichan* -- MIDI channel (1-16)

*ictlnum1 ... ictlnum64* -- MIDI control number (0-127)

*imin1 ... imin64* -- minimum values for each controller

*imax1 ... imax64* -- maximum values for each controller

*init1 ... init64* -- initial value for each controller

*ifn1 ... ifn64* -- function table for conversion for each controller

*icutoff1 ... icutoff64* -- low-pass filter cutoff frequency for each controller

## Performance

*k1 ... k64* -- output values

*slider64* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider64* allows a bank of 64 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

In the i-rate version of *slider64*, there is not an initial value input argument, because the output is gotten directly from current status of internal controller array of Csound.

## See Also

*s16b14*, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64f* *slider8*, *slider8f*

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# slider64f

slider64f — Creates a bank of 64 different MIDI control message numbers, filtered before output.

## Description

Creates a bank of 64 different MIDI control message numbers, filtered before output.

## Syntax

```
k1,...,k64 slider64f ichan, ictlnum1, imin1, imax1, init1, ifn1, \  
            icutoff1,..., ictlnum64, imin64, imax64, init64, ifn64, icutoff64
```

## Initialization

*ichan* -- MIDI channel (1-16)

*ictlnum1* ... *ictlnum64* -- MIDI control number (0-127)

*imin1* ... *imin64* -- minimum values for each controller

*imax1* ... *imax64* -- maximum values for each controller

*init1* ... *init64* -- initial value for each controller

*ifn1* ... *ifn64* -- function table for conversion for each controller

*icutoff1* ... *icutoff64* -- low-pass filter cutoff frequency for each controller

## Performance

*k1* ... *k64* -- output values

*slider64f* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider64f* allows a bank of 64 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.



## Avertissement

*slider64f* opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

## See Also

*s16b14, s32b14, slider16, slider16f, slider32, slider32f, slider64, slider8, slider8f*

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# slider8

slider8 — Creates a bank of 8 different MIDI control message numbers.

## Description

Creates a bank of 8 different MIDI control message numbers.

## Syntax

```
i1,...,i8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum8, imin8, imax8, init8, ifn8  
  
k1,...,k8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum8, imin8, imax8, init8, ifn8
```

## Initialization

*i1 ... i64* -- output values

*ichan* -- MIDI channel (1-16)

*ictlnum1 ... ictlnum64* -- MIDI control number (0-127)

*imin1 ... imin64* -- minimum values for each controller

*imax1 ... imax64* -- maximum values for each controller

*init1 ... init64* -- initial value for each controller

*ifn1 ... ifn64* -- function table for conversion for each controller

*icutoff1 ... icutoff64* -- low-pass filter cutoff frequency for each controller

## Performance

*k1 ... k64* -- output values

*slider8* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider8* allows a bank of 8 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.



In the i-rate version of *slider8*, there is not an initial value input argument, because the output is gotten directly from current status of internal controller array of Csound.

## See Also

*s16b14*, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8f*

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# slider8f

slider8f — Creates a bank of 8 different MIDI control message numbers, filtered before output.

## Description

Creates a bank of 8 different MIDI control message numbers, filtered before output.

## Syntax

```
k1,...,k8 slider8f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \  
..., ictlnum8, imin8, imax8, init8, ifn8, icutoff8
```

## Initialization

*ichan* -- MIDI channel (1-16)

*ictlnum1* ... *ictlnum64* -- MIDI control number (0-127)

*imin1* ... *imin64* -- minimum values for each controller

*imax1* ... *imax64* -- maximum values for each controller

*init1* ... *init64* -- initial value for each controller

*ifn1* ... *ifn64* -- function table for conversion for each controller

*icutoff1* ... *icutoff64* -- low-pass filter cutoff frequency for each controller

## Performance

*k1* ... *k64* -- output values

*slider8f* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider8f* allows a bank of 8 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using '\' (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.



## Avertissement

*slider8f* opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

## See Also

*s16b14*, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*

## Credits

Author: Gabriel Maldonado  
Italy  
December 1998

New in Csound version 3.50

Thanks goes to Rasmus Ekman for pointing out the correct MIDI channel and controller number ranges.

# slider16table

slider16table — Stores a bank of 16 different MIDI control messages to a table.

## Description

Stores a bank of 16 different MIDI control messages to a table.

## Syntax

```
kflag slider16table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1, ...., ictlnum16, imin16, imax16, init16, ifn16
```

## Initialization

*i1 ... i16* -- output values

*ichan* -- MIDI channel (1-16)

*ioutTable* -- number of the table that will contain the output

*ioffset* -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

*ictlnum1 ... ictlnum16* -- MIDI control number (0-127)

*imin1 ... imin16* -- minimum values for each controller

*imax1 ... imax16* -- maximum values for each controller

*init1 ... init16* -- initial value for each controller

*ifn1 ... ifn16* -- function table for conversion for each controller

## Performance

*kflag* -- a flag that informs if any control-change message in the bank has been received. In this case *kflag* is set to 1 is set to 1. Otherwise is set to zero.

*slider16table* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider16table* allows a bank of 16 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using

the separate ones (*ctrl7* and *tonek*) when more controllers are required.

*slider16table* is very similar to *slider16* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in *sliderNtable(f)*, you have to set a linear response and a minimum of zero and a maximum of one in all sliders.

## See Also

*slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slidertable8*, *slider8tablef*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# slider16tablef

slider16tablef — Stores a bank of 16 different MIDI control messages to a table, filtered before output.

## Description

Stores a bank of 16 different MIDI control messages to a table, filtered before output.

## Syntax

```
kflag slider16tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1, icutoff1, .... , ictlnum16, imin16, imax16, init16, ifn16, icutoff16
```

## Initialization

*ichan* -- MIDI channel (1-16)

*ioffset* -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

*ioutTable* -- number of the table that will contain the output

*ictlnum1* ... *ictlnum16* -- MIDI control number (0-127)

*imin1* ... *imin16* -- minimum values for each controller

*imax1* ... *imax16* -- maximum values for each controller

*init1* ... *init16* -- initial value for each controller

*ifn1* ... *ifn16* -- function table for conversion for each controller

*icutoff1* ... *icutoff16* -- low-pass filter cutoff frequency for each controller

## Performance

*kflag* -- a flag that informs if any control-change message in the bank has been received. In this case *kflag* is set to 1 is set to 1. Otherwise is set to zero.

*slider16tablef* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider16tablef* allows a bank of 16 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

*slider8table* is very similar to *slider16tablef* and *sliderNf* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in *sliderNtable(f)*, you have to set a linear response and a minimum of zero and a maximum of one in all sliders.



### Avertissement

*slider16tablef* does not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

## See Also

*slider16table*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*, *slider8tablef*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# slider32table

slider32table — Stores a bank of 32 different MIDI control messages to a table.

## Description

Creates a bank of 32 different MIDI control messages to a table.

## Syntax

```
kflag slider32table ichan, ioutTable, ioffset, ictlnum1, imin1, \  
imax1, init1, ifn1, ... , ictlnum32, imin32, imax32, init32, ifn32
```

## Initialization

*i1 ... i32* -- output values

*ichan* -- MIDI channel (1-16)

*ioutTable* -- number of the table that will contain the output

*ioffset* -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

*ictlnum1 ... ictlnum32* -- MIDI control number (0-127)

*imin1 ... imin32* -- minimum values for each controller

*imax1 ... imax32* -- maximum values for each controller

*init1 ... init32* -- initial value for each controller

*ifn1 ... ifn32* -- function table for conversion for each controller

## Performance

*kflag* -- a flag that informs if any control-change message in the bank has been received. In this case *kflag* is set to 1 is set to 1. Otherwise is set to zero.

*slider32table* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider32table* allows a bank of 32 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using



the separate ones (*ctrl7* and *tonek*) when more controllers are required.

*slider32table* is very similar to *slider32* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderN-table(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in *sliderNtable(f)*, you have to set a linear response and a minimum of zero and a maximum of one in all sliders.

## See Also

*slider16table*, *slider16tablef*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*, *slider8tablef*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# slider32tablef

slider32tablef — Creates a bank of 32 different MIDI control message numbers, filtered before output.

## Description

Creates a bank of 32 different MIDI control message numbers, filtered before output.

## Syntax

```
kflag slider32tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1, icutoff1, .... , ictlnum32, imin32, imax32, init32, ifn32, icutoff32
```

## Initialization

*ichan* -- MIDI channel (1-16)

*ioutTable* -- number of the table that will contain the output

*ioffset* -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

*ictlnum1* ... *ictlnum32* -- MIDI control number (0-127)

*imin1* ... *imin32* -- minimum values for each controller

*imax1* ... *imax32* -- maximum values for each controller

*init1* ... *init32* -- initial value for each controller

*ifn1* ... *ifn32* -- function table for conversion for each controller

*icutoff1* ... *icutoff32* -- low-pass filter cutoff frequency for each controller

## Performance

*kflag* -- a flag that informs if any control-change message in the bank has been received. In this case *kflag* is set to 1 is set to 1. Otherwise is set to zero.

*slider32tablef* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider32tablef* allows a bank of 32 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

*slider32tablef* is very similar to *slider32tablef* and *sliderNf* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in *sliderNtable(f)*, you have to set a linear response and a minimum of zero and a maximum of one in all sliders.



### Avertissement

*slider32tablef* opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

## See Also

*slider16*, *slider16f*, *slider32*, *slider64*, *slider64f*, *slider8*, *slider8f*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# slider64table

slider64table — Stores a bank of 64 different MIDI control messages to a table.

## Description

Creates a bank of 64 different MIDI control messages to a table.

## Syntax

```
kflag slider64table ichan, ioutTable, ioffset, ictlnum1, imin1, \  
imax1, init1, ifn1, ...., ictlnum64, imin64, imax64, init64, ifn64
```

## Initialization

*i1 ... i64* -- output values

*ichan* -- MIDI channel (1-16)

*ioutTable* -- number of the table that will contain the output

*ioffset* -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

*ictlnum1 ... ictlnum64* -- MIDI control number (0-127)

*imin1 ... imin64* -- minimum values for each controller

*imax1 ... imax64* -- maximum values for each controller

*init1 ... init64* -- initial value for each controller

*ifn1 ... ifn64* -- function table for conversion for each controller

## Performance

*kflag* -- a flag that informs if any control-change message in the bank has been received. In this case *kflag* is set to 1 is set to 1. Otherwise is set to zero.

*slider64table* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider64table* allows a bank of 64 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using

the separate ones (*ctrl7* and *tonek*) when more controllers are required.

*slider64table* is very similar to *slider64* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderN-table(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in *sliderNtable(f)*, you have to set a linear response and a minimum of zero and a maximum of one in all sliders.

## See Also

*slider16table*, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64tablef* *slider8table*, *slider8tablef*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# slider64tablef

slider64tablef — Stores a bank of 64 different MIDI control messages to a table, filtered before output.

## Description

Stores a bank of 64 different MIDI MIDI control messages to a table, filtered before output.

## Syntax

```
kflag slider64tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1, icutoff1, .... , ictlnum64, imin64, imax64, init64, ifn64, icutoff64
```

## Initialization

*ichan* -- MIDI channel (1-16)

*ioutTable* -- number of the table that will contain the output

*ioffset* -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

*ictlnum1* ... *ictlnum64* -- MIDI control number (0-127)

*imin1* ... *imin64* -- minimum values for each controller

*imax1* ... *imax64* -- maximum values for each controller

*init1* ... *init64* -- initial value for each controller

*ifn1* ... *ifn64* -- function table for conversion for each controller

*icutoff1* ... *icutoff64* -- low-pass filter cutoff frequency for each controller

## Performance

*kflag* -- a flag that informs if any control-change message in the bank has been received. In this case *kflag* is set to 1 is set to 1. Otherwise is set to zero.

*slider64tablef* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider64tablef* allows a bank of 64 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

*slider64tablef* is very similar to *slider64tablef* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in *sliderNtable(f)*, you have to set a linear response and a minimum of zero and a maximum of one in all sliders.



### Avertissement

*slider64tablef* opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

## See Also

*slider16table*, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider8table*, *slider8tablef*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# slider8table

slider8table — Stores a bank of 8 different MIDI control messages to a table.

## Description

Stores a bank of 8 different MIDI control messages to a table.

## Syntax

```
kflag slider8table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1,..., ictlnum8, imin8, imax8, init8, ifn8
```

## Initialization

*i1 ... i8* -- output values

*ichan* -- MIDI channel (1-16)

*ioutTable* -- number of the table that will contain the output

*ioffset* -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

*ictlnum1 ... ictlnum8* -- MIDI control number (0-127)

*imin1 ... imin8* -- minimum values for each controller

*imax1 ... imax8* -- maximum values for each controller

*init1 ... init8* -- initial value for each controller

*ifn1 ... ifn8* -- function table for conversion for each controller

## Performance

*kflag* -- a flag that informs if any control-change message in the bank has been received. In this case *kflag* is set to 1 is set to 1. Otherwise is set to zero.

*slider8table* handles a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider8table* allows a bank of 8 different MIDI control message numbers.

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using



the separate ones (*ctrl7* and *tonek*) when more controllers are required.

*slider8table* is very similar to *slider8* and *sliderN* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderN-table(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in *sliderNtable(f)*, you have to set a linear response and a minimum of zero and a maximum of one in all sliders.

## See Also

*slider16table*, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8tabletablef*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# slider8tablef

slider8tablef — Stores a bank of 8 different MIDI control messages to a table, filtered before output.

## Description

Stores a bank of 8 different MIDI control messages to a table, filtered before output.

## Syntax

```
kflag slider8tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum8, imin8, imax8, init8, ifn8, icutoff8
```

## Initialization

*ichan* -- MIDI channel (1-16)

*ioutTable* -- number of the table that will contain the output

*ioffset* -- output table offset. A zero means that the output of the first slider will affect the first table element. A 10 means that the output of the first slider will affect the 11th table element.

*ictlnum1* ... *ictlnum8* -- MIDI control number (0-127)

*imin1* ... *imin8* -- minimum values for each controller

*imax1* ... *imax8* -- maximum values for each controller

*init1* ... *init8* -- initial value for each controller

*ifn1* ... *ifn8* -- function table for conversion for each controller

*icutoff1* ... *icutoff8* -- low-pass filter cutoff frequency for each controller

## Performance

*kflag* -- a flag that informs if any control-change message in the bank has been received. In this case *kflag* is set to 1 is set to 1. Otherwise is set to zero.

*slider8tablef* is a bank of MIDI controllers, useful when using MIDI mixer such as Kawai MM-16 or others for changing whatever sound parameter in real-time. The raw MIDI control messages at the input port are converted to agree with *iminN* and *imaxN*, and an initial value can be set. Also, an optional non-interpolated function table with a custom translation curve is allowed, useful for enabling exponential response curves.

When no function table translation is required, set the *ifnN* value to 0, else set *ifnN* to a valid function table number. When table translation is enabled (i.e. setting *ifnN* value to a non-zero number referring to an already allocated function table), *initN* value should be set equal to *iminN* or *imaxN* value, else the initial output value will not be the same as specified in *initN* argument.

*slider8tablef* allows a bank of 8 different MIDI control message numbers. It filters the signal before output. This eliminates discontinuities due to the low resolution of the MIDI (7 bit). The cutoff frequency can be set separately for each controller (suggested range: .1 to 5 Hz).

As the input and output arguments are many, you can split the line using `\` (backslash) character (new in 3.47 version) to improve the readability. Using these opcodes is considerably more efficient than using the separate ones (*ctrl7* and *tonek*) when more controllers are required.

*slider8tablef* is very similar to *slider8f* and *sliderNf* family of opcodes (see their manual for more information). The actual difference is that the output is not stored to k-rate variables, but to a table, denoted by the *ioutTable* argument. It is possible to define a starting index in order to use the same table for more than one spider bank (or other purposes).

It is possible to use this opcode together with *FLslidBnk2Setk* and *FLslidBnk2*, so you can synchronize the position of the MIDI values to the position of the FLTK valuator widgets of *FLslidBnk2*. Notice that you have to specify the same min/max values as well the linear/exponential responses in both *sliderNtable(f)* and *FLslidBnk2*. The exception is when using table-indexed response instead of a lin/exp response. In this case, in order to achieve a useful result, the table-indexed response and actual min/max values must be set only in *FLslidBnk2*, whereas, in *sliderNtable(f)*, you have to set a linear response and a minimum of zero and a maximum of one in all sliders.



### Avertissement

*slider8tablef* opcodes do not output the required initial value immediately, but only after some k-cycles because the filter slightly delays the output.

## See Also

*slider16table*, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# sliderKawai

sliderKawai — Creates a bank of 16 different MIDI control message numbers from a KAWAI MM-16 midi mixer.

## Description

Creates a bank of 16 different MIDI control message numbers from a KAWAI MM-16 midi mixer.

## Syntax

```
k1, k2, ..., k16 sliderKawai imin1, imax1, init1, ifn1, \  
imin2, imax2, init2, ifn2, ..., imin16, imax16, init16, ifn16
```

## Initialization

*ictnum1* ... *ictnum32* -- MIDI control number (0-127)

*imin1* ... *imin16* -- minimum values for each controller

*imax1* ... *imax16* -- maximum values for each controller

*init1* ... *init16* -- initial value for each controller

*ifn1* ... *ifn16* -- function table for conversion for each controller

## Performance

*k1* ... *k16* -- output values

The opcode *sliderKawai* is equivalent to *slider16*, but it has the controller and channel numbers (*ichan* and *ictnum*) hard-coded to make for quick compatibility with the KAWAI MM-16 midi mixer. This device doesn't allow changing the midi message associated to each slider. It can only output on control 7 for each fader on a separate midi channel. This opcode is a quick way of assigning the mixer's 16 faders to k-rate variables in csound.

## See Also

*slider16*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

## Credits

Author: Gabriel Maldonado

New in Csound version 5.06

# sndload

sndload — Charge un fichier son en mémoire pour être utilisé par *loscilx*

## Description

*sndload* charge un fichier son en mémoire pour être utilisé par *loscilx*.

## Syntaxe

```
sndload Sfname[, ifmt[, ichns[, isr[, ibas[, iamp[, istrtrt \
[, ilpmod[, ilps[, ilpe]]]]]]]]]
```

## Initialisation

*Sfname* - nom du fichier sous la forme d'une constante, d'une variable ou d'un p-champ chaîne de caractères, ou bien un nombre utilisé comme index dans un ensemble de chaînes de caractères avec *strset* ou, s'il n'y a pas de chaîne disponible, pour générer un nom de fichier au format *soundin.n*. Si le nom de fichier ne comprend pas un chemin complet, le fichier est d'abord cherché dans le répertoire courant, puis dans celui qui est spécifié par *SSDIR* (si défini), et finalement par *SFDIR*. Si le même fichier a déjà été chargé antérieurement, il n'est pas relu, mais les paramètres *ibas*, *iamp*, *istrtrt*, *ilpmod*, *ilps* et *ilpe* sont quand même mis à jour.

*ifmt* (facultatif, zéro par défaut) - format d'échantillon par défaut pour les fichiers son bruts (sans en-tête) ; si le fichier a un en-tête, cet argument est ignoré. Les valeurs possibles sont :

- 1 : interdit les fichiers sans en-tête (échec avec une erreur d'initialisation)
- 0 : utilise le format spécifié dans la ligne de commande
- 1 : entiers signés sur 8 bit
- 2 : a-law
- 3 : u-law
- 4 : entiers signés sur 16 bit
- 5 : entiers signés sur 32 bit
- 6 : flottants sur 32 bit
- 7 : entiers non signés sur 8 bit
- 8 : entiers signés sur 24 bit
- 9 : flottants sur 64 bit

*ichns* (facultatif, zéro par défaut) - nombre de canaux par défaut pour les fichiers son bruts (sans en-tête) ; si le fichier a un en-tête, cet argument est ignoré. Les valeurs nulle ou négatives sont interprétées comme 1 canal.

*isr* (facultatif, zéro par défaut) - taux d'échantillonnage par défaut pour les fichiers son bruts (sans en-tête) ; si le fichier a un en-tête, cet argument est ignoré. Les valeurs nulle ou négatives sont interprétées comme le taux d'échantillonnage de l'orchestre (*sr*).

*ibas* (facultatif, zéro par défaut) - fréquence de base en Hz. Si elle est positive, elle remplace la valeur spécifiée dans l'en-tête du fichier son ; sinon, la valeur de l'en-tête est utilisée si elle est présente, et 1.0 si le fichier ne contient pas cette information.

*iamp* (facultatif, zéro par défaut) - pondération de l'amplitude. Si elle est différente de zéro, elle remplace la valeur spécifiée dans l'en-tête du fichier son (note : les valeurs négatives sont permises, elles inversent la phase de la sortie) ; sinon, la valeur de l'en-tête est utilisée si elle est présente, et 1.0 si le fichier ne contient pas cette information.

*istrt* (facultatif, -1 par défaut) - position du début en trames d'échantillon, peut être fractionnaire. Si elle est non négative, elle remplace la valeur spécifiée dans l'en-tête du fichier son ; sinon, la valeur de l'en-tête est utilisée si elle est présente, et 0 si le fichier ne contient pas cette information. Note : même si cet argument est spécifié, le fichier entier est lu en mémoire.

*ilpm* (facultatif, -1 par défaut) - mode de boucle, l'un des suivants :

n'importe quelle valeur négative : utilise l'information de boucle spécifiée dans l'en-tête du fichier son, ignorant *ilps* et *ilpe*

0 : pas de boucle (*ilps* et *ilpe* sont ignorés)

1 : boucle à l'endroit (cycle autour de la fin de boucle si elle est traversée en avançant, et cycle autour du début du boucle s'il est traversé en reculant)

2 : boucle à l'envers (change de direction à la fin de boucle si elle est traversée en avançant, et cycle autour du début de boucle s'il est traversé en reculant)

3 : boucle à l'endroit et à l'envers (change de direction aux deux points de boucle s'ils sont traversés comme décrit ci-dessus)

*ilps* (facultatif, zéro par défaut) - début de boucle en trames d'échantillon (valeurs fractionnaires autorisées), ou fin de boucle si *ilps* est supérieur à *ilpe*. Ignoré sauf si *ilpm* vaut 1, 2 ou 3. Si les points de boucle sont égaux, la boucle se fait sur l'échantillon complet.

*ilpe* (facultatif, zéro par défaut) - fin de boucle en trames d'échantillon (valeurs fractionnaires autorisées), ou début de boucle si *ilps* est supérieur à *ilpe*. Ignoré sauf si *ilpm* vaut 1, 2 ou 3. Si les points de boucle sont égaux, la boucle se fait sur l'échantillon complet.

## Crédits

Ecrit par Istvan Varga.

2006

Nouveau dans Csound 5.03

# sndloop

sndloop — Une boucle de son avec contrôle de la hauteur.

## Description

Cet opcode enregistre l'entrée audio et la restitue dans une boucle avec une durée définie par l'utilisateur et un fondu enchainé. On peut également contrôler la hauteur de la boucle et sa lecture à l'envers.

## Syntaxe

```
asig, krec sndloop ain, kpitch, ktrig, idur, ifad
```

## Initialisation

*idur* -- durée de la boucle en secondes.

*ifad* -- durée du fondu enchainé en secondes.

## Exécution

*asig* -- signal de sortie.

*krec* -- signal d'activation de l'enregistrement, 1 lors de l'enregistrement, 0 sinon.

*kpitch* -- contrôle de la hauteur (rapport de transposition) ; avec des valeurs négatives, la boucle est jouée à l'envers.

*ktrig* -- signal de déclenchement : lorsqu'il vaut 0, le traitement est suspendu. Lorsqu'il change (*ktrig* >= 1), l'opcode commence à enregistrer jusqu'à ce que la mémoire de la boucle soit pleine. Puis il restitue ensuite le son en boucle jusqu'au prochain changement (*ktrig* = 0). Un autre enregistrement peut recommencer lorsque *ktrig* >= 1.

## Exemples

### Exemple 496. Exemple

```
asig      in           ; get the signal in
ktrig     line         ; trigger signal
aout,krec sndloop asig, 1, ktrig, 4, 0.05 ; rec starts at 1 sec, for 4 secs 0.05 crossfade
          printk      ; prints the recording signal
          out          aout
```

L'exemple ci-dessus montre l'opération de base de *sndloop*. La hauteur peut-être contrôlée au taux-k, l'enregistrement commence dès que le signal de déclenchement est >= 1. L'enregistrement peut recommencer en fixant la valeur du signal de déclenchement à 0 puis de nouveau à 1.

## Crédits

Auteur : Victor Lazzarini  
Avril 2005

Nouveau dans la version 5.00



# sndwarp

*sndwarp* — Lit un son mono échantillonné dans une table et lui applique une modification de durée et/ou de hauteur.

## Description

*sndwarp* lit des échantillons sonores dans une table et applique une modification de durée et/ou de hauteur. Les modifications du temps et de la fréquence sont indépendantes l'une de l'autre. Par exemple un son peut être ralenti en durée tout en étant transposé dans l'aigu !

Les arguments de taille de fenêtre et de chevauchement influent grandement sur le résultat et seront fixés par expérimentation. En général ils doivent être aussi petits que possible. Par exemple, on peut commencer avec *iwsz*=*sr*/10 et *ioverlap*=15. Essayer *irandw*=*iwsz*\*0,2. Si l'on peut arriver à ses fins avec moins de chevauchements, le programme sera plus rapide. Mais si ces dernières sont en nombre insuffisant, on peut entendre des fluctuations d'amplitude. L'algorithme réagit différemment selon le son en entrée et il n'y a pas de règle fixe adaptée à toutes les circonstances. Si l'on arrive à trouver les bons réglages, on peut obtenir d'excellents résultats.

## Syntaxe

```
ares [, ac] sndwarp xamp, xtimewarp, xresample, ifn1, ibeg, iwsz, \
    irandw, ioverlap, ifn2, itimemode
```

## Initialisation

*ifn1* -- le numéro de la table contenant les échantillons qui seront traités par *sndwarp*. *GEN01* est le générateur de fonction approprié pour mémoriser les échantillons d'un fichier son pré-existant.

*ibeg* -- le temps en secondes à partir duquel commencera la lecture dans la table. Lorsque *itimemode* est différent de zéro, la valeur de *xtimewarp* est décalée de *ibeg*.

*iwsz* -- la taille en échantillons de la fenêtre utilisée dans l'algorithme de variation de la durée.

*irandw* -- la largeur de bande d'un générateur de nombres aléatoires. Les nombres aléatoires seront ajoutés à *iwsz*.

*ioverlap* -- détermine la densité de fenêtres se chevauchant.

*ifn2* -- une fonction qui fournit la forme de la fenêtre. On l'utilise habituellement pour créer une sorte de rampe qui part de zéro au début et qui y retourne à la fin de chaque fenêtre. Essayer d'utiliser une moitié de sinusoïde (c-à-d : f1 0 16384 9 .5 1 0) qui fonctionne plutôt bien. On peut utiliser d'autres formes.

## Exécution

*ares* -- l'unique canal de sortie du générateur unitaire *sndwarp*. *sndwarp* suppose que la table de fonction contenant le signal échantillonné est monophonique. *sndwarp* indexera la table avec un incrément d'un seul échantillon. Il faut ainsi remarquer que si l'on utilise un signal stéréo avec *sndwarp*, la durée et la hauteur seront altérées en conséquence.

*ac* (facultatif) -- une version mono-couche (pas de superpositions), et non fenêtrée du signal modifié en durée et/ou en hauteur. Elle est fournie afin de permettre de pondérer l'amplitude du signal de sortie, qui contient habituellement beaucoup de versions se chevauchant et fenêtrées du signal, avec une version

épurée du signal modifié en durée et en hauteur. Le traitement de *sndwarp* peut causer des variations notables en amplitude (en plus ou en moins), à cause de la différence de temps entre les superpositions lorsque la variation de durée est appliquée. Si on l'utilise avec une unité *balance*, *ac* permet d'améliorer grandement la qualité sonore.

*xamp* -- la valeur qui sert à pondérer l'amplitude (voir la note sur son utilisation avec *ac*).

*xtimewarp* -- détermine comment la durée du signal en entrée sera allongée ou raccourcie. Il y a deux manières d'utiliser cet argument selon la valeur donnée à *itimemode*. Si la valeur de *itimemode* est 0, *xtimewarp* changera l'échelle temporelle du son. Par exemple, une valeur de 2 doublera la durée du son. Si *itimemode* a une valeur non nulle, alors *xtimewarp* est utilisé comme un pointeur temporel de la même manière que dans *lpread* et dans *pvoc*. Un des exemples ci-dessous illustre cette possibilité. Dans les deux cas, la hauteur ne sera *pas* altérée par le traitement. La transposition de hauteur est effectuée indépendamment au moyen de *xresample*.

*xresample* -- le facteur de changement de la hauteur du son. Par exemple, une valeur de 2 produira un son une octave plus haut que l'original. La durée du son, quant à elle, ne sera *pas* modifiée.

## Exemples

Voici en exemple de l'opcode *sndwarp*. Il utilise les fichiers *sndwarp.csd* [examples/sndwarp.csd] et *mary.wav* [examples/mary.wav].

### Exemple 497. Exemple de l'opcode *sndwarp*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o sndwarp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
; Use the audio file defined in Table #1.
a1 loscil 30000, 1, 1, 1

out a1
endin

; Instrument #2 - time-stretch an audio file.
instr 2
kamp init 6500
; Start at 1 second and end at 3.5 seconds.
ktimewarp line 1, p3, 3.5
; Playback at the normal speed.
kresample init 1
; Use the audio file defined in Table #1.
ifn1 = 1
ibeg = 0
iwsz = 4410
irandw = 882
ioverlap = 15
; Use Table #2 for the windowing function.
ifn2 = 2
```

```

; Use the ktimewarp parameter as a "time" pointer.
itimemode = 1

al sndwarp kamp, ktimewarp, kresample, ifn1, ibeg, iwsiz, irandw, ioverlap, ifn2, itimemode
out al
endin

</CsInstruments>
<CsScore>

; Table #1: an audio file.
f 1 0 262144 1 "mary.wav" 0 0 0
; Table #2: half of a sine wave.
f 2 0 16384 9 0.5 1 0

; Play Instrument #1 for 3.5 seconds.
i 1 0 3.5
; Play Instrument #2 for 7 seconds (time-stretched).
i 2 3.5 10.5
e

</CsScore>
</CsoundSynthesizer>

```

L'exemple ci-dessous montre un ralentissement du son stocké dans la table (*ifn1*). Pendant toute la durée de la note, le ralentissement s'intensifiera depuis l'original jusqu'à un son dix fois plus « lent » que l'original. Pendant ce temps, la hauteur montera progressivement d'une octave.

```

iwindfun = 1
isampfun = 2
ibeg = 0
iwindsize = 2000
iwindrand = 400
ioverlap = 10
awarp line 1, p3, 1
aresamp line 1, p3, 2
kenv line 1, p3, .1
asig sndwarp kenv, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, 0

```

Voici maintenant un exemple utilisant *xtimewarp* comme pointeur temporel et la stéréophonie :

```

itimemode = 1
atime line 0, p3, 10
ar1, ar2 sndwarpst kenv, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
iwindfun, itimemode

```

Ci-dessus, *atime* avance le pointeur temporel utilisé dans *sndwarpst* de 0 à 10 sur toute la durée de la note. Si *p3* vaut 20 alors le son sera deux fois plus lent que l'original. Bien sûr, on peut utiliser une fonction plus complexe qu'une simple ligne droite pour contrôler le facteur temporel.

Maintenant le même exemple que ci-dessus mais en utilisant la fonction *balance* avec les sorties facultatives :

```

asig, acmp sndwarp 1, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, itimemode
abal balance asig, acmp

asig1, asig2, acmp1, acmp2 sndwarpst 1, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
iwindfun, itimemode
abal1 balance asig1, acmp1
abal2 balance asig2, acmp2

```

Noter l'utilisation de l'unité *balance* dans les deux exemples ci-dessus. La sortie de *balance* peut ensuite être pondérée, enveloppée, envoyée à un *out* ou un *outs*, etc. Noter que les arguments d'amplitude de *sndwarp* et de *sndwarpst* valent « 1 » dans ces exemples. En pondérant le signal après son traitement par *sndwarp*, *abal*, *abal1*, et *abal2* contiendront des signaux ayant à peu près la même amplitude que le signal original traité par *sndwarp*. Il est ainsi plus facile de prédire les niveaux et d'éviter d'avoir des échantillons hors intervalle ou des valeurs d'échantillon trop petites.



### Conseil Supplémentaire

N'utilisez la version stéréo que si vous avez réellement besoin de traiter un fichier stéréo. Elle est sensiblement plus lente que la version mono et si vous utilisez la fonction *balance*, c'est encore plus lent. Il n'y a aucun inconvénient à utiliser un *sndwarp* mono dans un orchestre stéréo puis d'envoyer le résultat à un ou aux deux canaux de la sortie stéréo.

## Voir Aussi

*sndwarpst*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1997

Exemple écrit par Kevin Conder.

# sndwarpst

sndwarpst — Lit un son stéréo échantillonné dans une table et lui applique une modification de durée et/ou de hauteur.

## Description

*sndwarpst* lit des échantillons stéréo sonores dans une table et applique une modification de durée et/ou de hauteur. Les modifications du temps et de la fréquence sont indépendantes l'une de l'autre. Par exemple un son peut être ralenti en durée tout en étant transposé dans l'aigu !

Les arguments de taille de fenêtre et de chevauchement influent grandement sur le résultat et seront fixés par expérimentation. En général ils doivent être aussi petits que possible. Par exemple, on peut commencer avec  $iwsz = sr/10$  et  $ioverlap = 15$ . Essayer  $irandw = iwsz * 0.2$ . Si l'on peut arriver à ses fins avec moins de chevauchements, le programme sera plus rapide. Mais si ces dernières sont en nombre insuffisant, on peut entendre des fluctuations d'amplitude. L'algorithme réagit différemment selon le son en entrée et il n'y a pas de règle fixe adaptée à toutes les circonstances. Si l'on arrive à trouver les bons réglages, on peut obtenir d'excellents résultats.

## Syntaxe

```
ar1, ar2 [,ac1] [, ac2] sndwarpst xamp, xtimewarp, xresample, ifn1, \
ibeg, iwsz, irandw, ioverlap, ifn2, itimemode
```

## Initialisation

*ifn1* -- le numéro de la table contenant les échantillons qui seront traités par *sndwarpst*. *GEN01* est le générateur de fonction approprié pour mémoriser les échantillons d'un fichier son pré-existant.

*ibeg* -- le temps en secondes à partir duquel commencera la lecture dans la table. Lorsque *itimemode* est différent de zéro, la valeur de *xtimewarp* est décalée de *ibeg*.

*iwsz* -- la taille en échantillons de la fenêtre utilisée dans l'algorithme de variation de la durée.

*irandw* -- la largeur de bande d'un générateur de nombres aléatoires. Les nombres aléatoires seront ajoutés à *iwsz*.

*ioverlap* -- détermine la densité de fenêtres se chevauchant.

*ifn2* -- une fonction qui fournit la forme de la fenêtre. On l'utilise habituellement pour créer une sorte de rampe qui part de zéro au début et qui y retourne à la fin de chaque fenêtre. Essayer d'utiliser une moitié de sinuséide (c-à-d : f1 0 16384 9 .5 1 0) qui fonctionne plutôt bien. On peut utiliser d'autres formes.

## Exécution

*ar1, ar2* -- *ar1* et *ar2* sont les sorties stéréo (gauche et droite) de *sndwarpst*. *sndwarpst* suppose que la table de fonction contenant le signal échantillonné est stéréophonique. *sndwarpst* indexera la table avec un incrément de deux échantillons. Il faut ainsi remarquer que si l'on utilise un signal mono avec *sndwarpst*, la durée et la hauteur seront altérées en conséquence.

*ac1, ac2* -- *ac1* et *ac2* sont des versions mono-couche (pas de superpositions), et non fenêtrées du signal modifié en durée et/ou en hauteur. Elles sont fournies afin de permettre de pondérer l'amplitude du signal de sortie, qui contient habituellement beaucoup de versions se chevauchant et fenêtrées du signal,

avec une version épurée du signal modifié en durée et en hauteur. Le traitement de *sndwarpst* peut causer des variations notables en amplitude (en plus ou en moins), à cause de la différence de temps entre les superpositions lorsque la variation de durée est appliquée. Si on les utilise avec une unité *balance*, *ac1* et *ac2* permettent d'améliorer grandement la qualité sonore. Ils sont facultatifs mais il faut noter que la syntaxe exige la présence des deux arguments (utiliser les deux ou aucun). Un exemple de leur utilisation est donné ci-dessous.

*xamp* -- la valeur qui sert à pondérer l'amplitude (voir la note sur son utilisation avec *ac1* et *ac2*).

*xtimewarp* -- détermine comment la durée du signal en entrée sera allongée ou raccourcie. Il y a deux manières d'utiliser cet argument selon la valeur donnée à *itimemode*. Si la valeur de *itimemode* est 0, *xtimewarp* changera l'échelle temporelle du son. Par exemple, une valeur de 2 doublera la durée du son. Si *itimemode* a une valeur non nulle, alors *xtimewarp* est utilisé comme un pointeur temporel de la même manière que dans *lpread* et dans *pvoc*. Un des exemples ci-dessous illustre cette possibilité. Dans les deux cas, la hauteur ne sera *pas* altérée par le traitement. La transposition de hauteur est effectuée indépendamment au moyen de *xresample*.

*xresample* -- le facteur de changement de la hauteur du son. Par exemple, une valeur de 2 produira un son une octave plus haut que l'original. La durée du son, quant à elle, ne sera *pas* modifiée.

## Exemples

L'exemple ci-dessous montre un ralentissement du son stocké dans la table (*ifn1*). Pendant toute la durée de la note, le ralentissement s'intensifiera depuis l'original jusqu'à un son dix fois plus « lent » que l'original. Pendant ce temps, la hauteur montera progressivement d'une octave.

```

iwindfun = 1
isampfun = 2
ibeg = 0
iwindsize = 2000
iwindrand = 400
ioverlap = 10
awarp line 1, p3, 1
aresamp line 1, p3, 2
kenv line 1, p3, .1
asig sndwarp kenv, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, 0

```

Voici maintenant un exemple utilisant *xtimewarp* comme pointeur temporel et la stéréophonie :

```

itimemode = 1
atime line 0, p3, 10
ar1, ar2 sndwarpst kenv, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
iwindfun, itimemode

```

Ci-dessus, *atime* avance le pointeur temporel utilisé dans *sndwarpst* de 0 à 10 sur toute la durée de la note. Si *p3* vaut 20 alors le son sera deux fois plus lent que l'original. Bien sûr, on peut utiliser une fonction plus complexe qu'une simple ligne droite pour contrôler le facteur temporel.

Maintenant le même exemple que ci-dessus mais en utilisant la fonction *balance* avec les sorties facultatives :

```

asig,acmp sndwarp 1, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, itimemode
abal balance asig, acmp
asig1,asig2,acmp1,acmp2 sndwarpst 1, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
iwindfun, itimemode
abal1 balance asig1, acmp1

```

```
abal2      balance asig2, acmp2
```

Noter l'utilisation de l'unité *balance* dans les deux exemples ci-dessus. La sortie de *balance* peut ensuite être pondérée, enveloppée, envoyée à un *out* ou un *outs*, etc. Noter que les arguments d'amplitude de *sndwarp* et de *sndwarpst* valent « 1 » dans ces exemples. En pondérant le signal après son traitement par *sndwarp*, *abal*, *abal1*, et *abal2* contiendront des signaux ayant à peu près la même amplitude que le signal original traité par *sndwarp*. Il est ainsi plus facile de prédire les niveaux et d'éviter d'avoir des échantillons hors intervalle ou des valeurs d'échantillon trop petites.



### Conseil Supplémentaire

N'utilisez la version stéréo que si vous avez réellement besoin de traiter un fichier stéréo. Elle est sensiblement plus lente que la version mono et si vous utilisez la fonction *balance*, c'est encore plus lent. Il n'y a aucun inconvénient à utiliser un *sndwarp* mono dans un orchestre stéréo puis d'envoyer le résultat à un ou aux deux canaux de la sortie stéréo.

## Voir Aussi

*sndwarp*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1997

# socksend

socksend — Sends data to other processes using the low-level UDP or TCP protocols

## Description

Transmits data directly using the UDP (socksend and socksends) or TCP (stsend) protocol onto a network. The data is not subject to any encoding or special routing. The socksends opcode send a stereo signal interleaved.

## Syntax

```
socksend asig, Sipaddr, ippor, ilength
```

```
socksends asigl, asigr, Sipaddr, ippor,  
            ilength
```

```
stsend asig, Sipaddr, ippor
```

## Initialization

*Sipaddr* -- a string that is the IP address of the receiver in standard 4-octet dotted form.

*ippor* -- the number of the port that is used for the communication.

*ilength* -- the length of the individual packets in UDP transmission. This number must be sufficiently small to fit a single MTU, which is set to the save value of 1456. In UDP transmissions the receiver needs to know this value

## Performance

*asig, asigl, asigr* -- audio data to be transmitted.

## Example

The example shows a simple sine wave being sent just once to a computer called "172.16.0.255", on port 7777 using UDP. Note that .255 is often used for broadcasting.

```
sr = 44100  
ksmps = 100  
nchnls = 1  
  
instr 1  
a1 oscil 20000,441,1  
  socksend a1, "172.16.0.255",7777, 200  
endin
```

## Credits



Author: John ffitch  
2006

# sockrecv

sockrecv — Receives data from other processes using the low-level UDP or TCP protocols

## Description

Receives directly using the UDP (sockrecv and sockrecvs) or TCP (strecv) protocol onto a network. The data is not subject to any encoding or special routing. The sockrecvs opcode receives a stereo signal interleaved.

## Syntax

```
asig sockrecv iport, ilength
```

```
asigl, asigr sockrecvs iport, ilength
```

```
asig strecv Sipaddr, iport
```

## Initialization

*Sipaddr* -- a string that is the IP address of the sender in standard 4-octet dotted form.

*iport* -- the number of the port that is used for the communication.

*ilength* -- the length of the individual packets in UDP transmission. This number must be sufficiently small to fit a single MTU, which is set to the save value of 1456. In UDP transmissions the sender and receiver needs agree on this value

## Performance

*asig, asigl, asigr* -- audio data to be received.

## Example

The example shows a mono signal being received on port 7777 using UDP.

```
sr = 44100
ksmps = 100
nchnls = 1

instr 1
a1 sockrecv      7777, 200
  out  a1
endin
```

## Credits

Author: John ffitth

2006

# soundin

soundin — Lit des données audio mono depuis un périphérique externe ou un flot.

## Description

Lit des données audio mono depuis un périphérique externe ou un flot. On peut lire jusqu'à 24 canaux.

## Syntaxe

```
ar1[, ar2[, ar3[, ... a24]]] soundin ifilcod [, iskptim] [, iformat] \  
[, iskipinit] [, ibufsize]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son source. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) puis par SFDIR. Voir aussi *GEN01*.

*iskptim* (facultatif, 0 par défaut) -- portion du son en entrée à ignorer, exprimée en secondes. La valeur par défaut est 0. A partir de Csound 5.00, cette valeur peut être négative ce qui ajoute un délai au lieu d'une portion à ignorer.

*iformat* (facultatif, 0 par défaut) -- spécifie le format des données audio du fichier :

- 1 = caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 2 = octets sur 8 bit A-law
- 3 = octets sur 8 bit U-law
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers non signés sur 8 bit (non disponible dans les versions de Csound antérieures à la 5.00)
- 8 = entiers sur 24 bit (non disponible dans les versions de Csound antérieures à la 5.00)
- 9 = doubles sur 64 bit (non disponible dans les versions de Csound antérieures à la 5.00)

*iskipinit* -- supprime toute initialisation s'il est non nul (vaut 0 par défaut). Fut introduit dans la version 4\_23f13 et dans csound5.

*ibufsize* -- taille du tampon en échantillons mono (pas en trames d'échantillons). N'est pas disponible dans les versions de Csound antérieures à la 5.00. La taille de tampon par défaut est 2048.

Si *iformat* = 0, il est déduit de l'en-tête du fichier, et s'il n'y a pas d'en-tête, de l'option de ligne de commande *-o* de Csound. La valeur par défaut est 0.

## Exécution

*soundin* est fonctionnellement un générateur audio dont le signal est dérivé d'un fichier pré-existant. Le nombre de canaux lus est contrôlé par le nombre de variables résultat, *a1*, *a2*, etc., qui doivent concorder avec ceux du fichier d'entrée. Un opcode *soundin* ouvre le fichier chaque fois que l'instrument le contenant est initialisé, puis il le ferme chaque fois que l'instrument est arrêté.

Il peut y avoir n'importe quel nombre d'opcodes *soundin* dans un instrument de l'orchestre. Plusieurs d'entre eux peuvent lire simultanément depuis le même fichier.



### Note pour les utilisateurs de Windows

Les utilisateurs de Windows utilisent normalement des anti-slash, « \ », pour spécifier les chemins de leurs fichiers. Par exemple un utilisateur de Windows pourra utiliser le chemin « c:\music\samples\loop001.wav ». Ceci pose problème car les anti-slash servent habituellement à spécifier des caractères spéciaux.

Pour spécifier correctement ce chemin dans Csound on peut utiliser :

- soit le *slash* : c:/music/samples/loop001.wav
- soit le *caractère spécial d'anti-slash*, « \\ » : c:\\music\\samples\\loop001.wav

## Exemples

Voici un exemple de l'opcode *soundin*. Il utilise les fichiers *soundin.csd* [examples/soundin.csd] et *beats.wav* [examples/beats.wav].

### Exemple 498. Exemple de l'opcode *soundin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o soundin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
  asig soundin "beats.wav"
  out asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
```

i 1 0 3  
e

</CsScore>  
</CsoundSynthesizer>

## Voir Aussi

*diskin, in, inh, ino, inq, ins*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

Exemple écrit par Kevin Conder.

Avertissement pour les utilisateurs de Windows ajouté par Kevin Conder, avril 2002

# soundout

soundout — Obsolète. Écrit la sortie audio dans un fichier sur disque.

## Description



### Note

L'utilisation de *soundout* est déconseillée. Il vaut mieux utiliser *fout*.

Écrit la sortie audio dans un fichier sur disque.

## Syntaxe

```
soundout  asigl, ifilcod [, iformat]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son destination. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) puis par SFDIR. Voir aussi *GEN01*.

*iformat* (facultatif, 0 par défaut) -- spécifie le format des données audio du fichier :

- 1 = caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 2 = octets sur 8 bit A-law
- 3 = octets sur 8 bit U-law
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit

Si *iformat* = 0, il est déduit de l'en-tête du fichier, et s'il n'y a pas d'en-tête, de l'option de ligne de commande *-o* de Csound. La valeur par défaut est 0.

## Exécution

*soundout* écrit la sortie audio dans un fichier sur disque.



### Note

Il est recommandé d'utiliser *fout* plutôt que *soundout*

## Voir Aussi

*fout, out, outh, outh, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2 soundouts*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997



# soundouts

soundouts — Obsolète. Ecrit la sortie audio dans un fichier sur disque.

## Description



### Note

L'utilisation de *soundouts* est déconseillée. Il vaut mieux utiliser *fout*.

Ecrit la sortie audio dans un fichier sur disque.

## Syntaxe

```
soundouts asigl, asigr, ifilcod [, iformat]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son destination. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) puis par SFDIR. Voir aussi *GEN01*.

*iformat* (facultatif, 0 par défaut) -- spécifie le format des données audio du fichier :

- 1 = caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit

Si *iformat* = 0, il est déduit de l'option de ligne de commande *-o* de Csound. La valeur par défaut est 0.

## Exécution

*soundouts* écrit la sortie audio stéréo dans un fichier sur disque au format brut (sans en-tête) et sans mise à l'échelle 0dbFS. L'intervalle d'amplitude attendu des signaux audio dépend du format d'échantillon choisi.



### Note

Il est recommandé d'utiliser *fout* plutôt que *soundouts*

## Voir Aussi

*out, outh, uto, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2 soundout*

## Crédits

Auteur : Istvan Varga

# space

space — Distributes an input signal among 4 channels using cartesian coordinates.

## Description

*space* takes an input signal and distributes it among 4 channels using Cartesian xy coordinates to calculate the balance of the outputs. The xy coordinates can be defined in a separate text file and accessed through a Function statement in the score using *Gen28*, or they can be specified using the optional *kx*, *ky* arguments. The advantages to the former are:

1. A graphic user interface can be used to draw and edit the trajectory through the Cartesian plane
2. The file format is in the form time1 X1 Y1 time2 X2 Y2 time3 X3 Y3 allowing the user to define a time-tagged trajectory

*space* then allows the user to specify a time pointer (much as is used for *pvoc*, *lpread* and some other units) to have detailed control over the final speed of movement.

## Syntax

```
a1, a2, a3, a4 space asig, ifn, ktime, kreverbsend, kx, ky
```

## Initialization

*ifn* -- number of the stored function created using *Gen28*. This function generator reads a text file which contains sets of three values representing the xy coordinates and a time-tag for when the signal should be placed at that location. The file should look like:

```
0   -1   1
1    1   1
2    4   4
2.1 -4  -4
3   10 -10
5  -40   0
```

If that file were named « move » then the *Gen28* call in the score would like:

```
f1 0 0 28 "move"
```

*Gen28* takes 0 as the size and automatically allocates memory. It creates values to 10 milliseconds of resolution. So in this case there will be 500 values created by interpolating X1 to X2 to X3 and so on, and Y1 to Y2 to Y3 and so on, over the appropriate number of values that are stored in the function table. In the above example, the sound will begin in the left front, over 1 second it will move to the right front, over another second it move further into the distance but still in the left front, then in just 1/10th of a second it moves to the left rear, a bit distant. Finally over the last .9 seconds the sound will move to the

right rear, moderately distant, and it comes to rest between the two left channels (due west!), quite distant. Since the values in the table are accessed through the use of a time-pointer in the *space* unit, the actual timing can be made to follow the file's timing exactly or it can be made to go faster or slower through the same trajectory. If you have access to the GUI that allows one to draw and edit the files, there is no need to create the text files manually. But as long as the file is ASCII and in the format shown above, it doesn't matter how it is made!



### Important

If *ifn* is 0, then *space* will take its values for the xy coordinates from *kx* and *ky*.

## Performance

The configuration of the xy coordinates in *space* places the signal in the following way:

- a1 is -1, 1
- a2 is 1, 1
- a3 is -1, -1
- a4 is 1, -1

This assumes a loudspeaker set up as a1 is left front, a2 is right front, a3 is left back, a4 is right back. Values greater than 1 will result in sounds being attenuated, as if in the distance. *space* considers the speakers to be at a distance of 1; smaller values of xy can be used, but *space* will not amplify the signal in this case. It will, however balance the signal so that it can sound as if it were within the 4 speaker *space*. x=0, y=1, will place the signal equally balanced between left and right front channels, x=y=0 will place the signal equally in all 4 channels, and so on. Although there must be 4 output signals from *space*, it can be used in a 2 channel orchestra. If the xy's are kept so that  $Y \geq 1$ , it should work well to do panning and fixed localization in a stereo field.

*asig* -- input audio signal.

*ktime* -- index into the table containing the xy coordinates. If used like:

```
ktime      line 0, 5, 5
a1, a2, a3, a4 space asig, 1, ktime, ...
```

with the file « move » described above, the speed of the signal's movement will be exactly as described in that file. However:

```
ktime      line 0, 10, 5
```

the signal will move at half the speed specified. Or in the case of:

```
ktime      line 5, 15, 0
```

the signal will move in the reverse direction as specified and 3 times slower! Finally:

`ktime`      *line* 2, 10, 3

will cause the signal to move only from the place specified in line 3 of the text file to the place specified in line 5 of the text file, and it will take 10 seconds to do it.

*kreverb**send* -- the percentage of the direct signal that will be factored along with the distance as derived from the XY coordinates to calculate signal amounts that can be sent to reverb units such as reverb, or reverb2.

*kx*, *ky* -- when *ifn* is 0, *space* and *spdist* will use these values as the XY coordinates to localize the signal.

## Examples

```
instr 1
  asig      ;some audio signal
  ktime      line 0, p3, p10
  a1, a2, a3, a4    space asig,1, ktime, .1
  ar1, ar2, ar3, ar4 spsend

  ga1 = ga1+ar1
  ga2 = ga2+ar2
  ga3 = ga3+ar3
  ga4 = ga4+ar4

                                outq a1, a2, a3, a4
endin

instr 99 ; reverb instrument

  a1 reverb2 ga1, 2.5, .5
  a2 reverb2 ga2, 2.5, .5
  a3 reverb2 ga3, 2.5, .5
  a4 reverb2 ga4, 2.5, .5

  outq a1, a2, a3, a4
  ga1=0
  ga2=0
  ga3=0
  ga4=0
endin
```

In the above example, the signal, *asig*, is moved according to the data in Function #1 indexed by *ktime*. *space* sends the appropriate amount of the signal internally to *spsend*. The outputs of the *spsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

*space* can be useful for quad and stereo panning as well as fixed placement of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field using xy values from the score instead of a function table.

```
instr 1
  ...
  a1, a2, a3, a4    space asig, 0, 0, .1, p4, p5
  ar1, ar2, ar3, ar4 spsend

  ga1=ga1+ar1
```

```
ga2=ga2+ar2
                                outs  a1, a2
endin

instr 99 ; reverb...
....
endin
```

A few notes: p4 and p5 are the X and Y values

```
;place the sound in the left speaker and near
il 0 1 -1 1
;place the sound in the right speaker and far
il 1 1 45 45
;place the sound equally between left and right and in the middle ground distance
il 2 1 0 12
e
```

The next example shows a simple intuitive use of the distance values returned by *spdist* to simulate Doppler shift.

```
ktime          line 0, p3, 10
kdist          spdist 1, ktime
kfreq = (ifreq * 340) / (340 + kdist)
asig          oscili iamp, kfreq, 1

a1, a2, a3, a4  space asig, 1, ktime, .1
ar1, ar2, ar3, ar4 spsend
```

The same function and time values are used for both *spdist* and *space*. This insures that the distance values used internally in the *space* unit will be the same as those returned by *spdist* to give the impression of a Doppler shift!

## See Also

*spdist*, *spsend*

## Credits

Author: Richard Karpen  
Seattle, WA USA  
1998

New in Csound version 3.48

# spat3d

spat3d — Positions the input sound in a 3D space and allows moving the sound at k-rate.

## Description

This opcode positions the input sound in a 3D space, with optional simulation of room acoustics, in various output formats. *spat3d* allows moving the sound at k-rate (this movement is interpolated internally to eliminate "zipper noise" if sr not equal to kr).

## Syntax

```
aW, aX, aY, aZ spat3d ain, kX, kY, kZ, idist, ift, imode, imdel, iovr [, istor]
```

## Initialization

*idist* -- For modes 0 to 3, *idist* is the unit circle distance in meters. For mode 4, *idist* is the distance between microphones.

The following formulas describe amplitude and delay as a function of sound source distance from microphone(s):

$$\text{amplitude} = 1 / (0.1 + \text{distance})$$

$$\text{delay} = \text{distance} / 340 \text{ (in seconds)}$$

Distance can be calculated as:

$$\text{distance} = \sqrt{iX^2 + iY^2 + iZ^2}$$

In Mode 4, distance can be calculated as:

$$\begin{aligned} \text{distance from left mic} &= \sqrt{(iX + idist/2)^2 + iY^2 + iZ^2} \\ \text{distance from right mic} &= \sqrt{(iX - idist/2)^2 + iY^2 + iZ^2} \end{aligned}$$

With *spat3d* the distance between the sound source and any microphone should be at least  $(340 * 18) / \text{sr}$  meters. Shorter distances will work, but may produce artifacts in some cases. There is no such limitation for *spat3di* and *spat3dt*.

Sudden changes or discontinuities in sound source location can result in pops or clicks. Very fast movement may also degrade quality.

*ift* -- Function table storing room parameters (for free field spatialization, set it to zero or negative).

Table size is 54. The values in the table are:

Room Parameter	Purpose
0	Early reflection recursion depth (0 is the sound source, 1 is the first reflection etc.) for spat3d and spat3di. The number of echoes for four walls (front, back, right, left) is: $N = (2 * R + 2) * R$ . If all six walls are enabled: $N = (((4 * R + 6) * R + 8) * R) / 3$
1	Late reflection recursion depth (used by spat3dt only). spat3dt skips early reflections and renders echoes up to this level. If early reflection depth is negative, spat3d and spat3di will output zero, while spat3dt will start rendering from the sound source.
2	imdel for spat3d. Overrides opcode parameter if non-negative.
3	irlen for spat3dt. Overrides opcode parameter if non-negative.
4	idist value. Overrides opcode parameter if $\geq 0$ .
5	Random seed (0 - 65535) -1 seeds from current time.
6 - 53	wall parameters (w = 6: ceil, w = 14: floor, w = 22: front, w = 30: back, w = 38: right, w = 46: left)
w + 0	Enable reflections from this wall (0: no, 1: yes)
w + 1	Wall distance from listener (in meters)
w + 2	Randomization of wall distance (0 - 1) (in units of $1 / (\text{wall distance})$ )
w + 3	Reflection level (-1 - 1)
w + 4	Parametric equalizer frequency in Hz.
w + 5	Parametric equalizer level (1.0: no filtering)
w + 6	Parametric equalizer Q (0.7071: no resonance)
w + 7	Parametric equalizer mode (0: peak EQ, 1: low shelf, 2: high shelf)

*imode* -- Output mode

- 0: B format with W output only (mono)

$aout = aW$

- 1: B format with W and Y output (stereo)

$aleft = aW + 0.7071 * aY$   
 $aright = aW - 0.7071 * aY$

- 2: B format with W, X, and Y output (2D). This can be converted to UHJ:



```

aWre, aWim    hilbert aW
aXre, aXim    hilbert aX
aYre, aYim    hilbert aY
aWXr  = 0.0928*aXre + 0.4699*aWre
aWXiYr = 0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aleft  = aWXr + aWXiYr
aright = aWXr - aWXiYr

```

- 3: B format with all outputs (3D)
- 4: Simulates a pair of microphones (stereo output)

```

aW    butterlp aW, ifreq ; recommended values for ifreq
aY    butterlp aY, ifreq ; are around 1000 Hz
aleft = aW + aX
aright = aY + aZ

```

Mode 0 is the cheapest to calculate, while mode 4 is the most expensive.

In Mode 4, The optional lowpass filters can change the frequency response depending on direction. For example, if the sound source is located left to the listener then the high frequencies are attenuated in the right channel and slightly increased in the left. This effect can be disabled by not using filters. You can also experiment with other filters (tone etc.) for better effect.

Note that mode 4 is most useful for listening with headphones, and is also more expensive to calculate than the B-format (0 to 3) modes. The *idist* parameter in this case sets the distance between left and right microphone; for headphones, values between 0.2 - 0.25 are recommended, although higher settings up to 0.4 may be used for wide stereo effects.

More information about B format can be found here:  
[http://www.york.ac.uk/inst/mustech/3d\\_audio/ambis2.htm](http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm)

*imdel* -- Maximum delay time for spat3d in seconds. This has to be longer than the delay time of the latest reflection (depends on room dimensions, sound source distance, and recursion depth; using this formula gives a safe (although somewhat overestimated) value:

$$\text{imdel} = (R + 1) * \sqrt{W*W + H*H + D*D} / 340.0$$

where R is the recursion depth, W, H, and D are the width, height, and depth of the room, respectively).

*iovr* -- Oversample ratio for spat3d (1 to 8). Setting it higher improves quality at the expense of memory and CPU usage. The recommended value is 2.

*istor* (optional, default=0) -- Skip initialization if non-zero (default: 0).

## Performance

*aW, aX, aY, aZ* -- Output signals

	mode 0	mode 1	mode 2	mode 3	mode 4
aW	W out	W out	W out	W out	left chn / low freq.
aX	0	0	X out	X out	left chn / high freq.
aY	0	Y out	Y out	Y out	right chn / low freq.
aZ	0	0	0	Z out	right chn / high fr.

*ain* -- Input signal

*kX*, *kY*, *kZ* -- Sound source coordinates (in meters)

If you encounter very slow performance (up to 100 times slower), it may be caused by denormals (this is also true of many other IIR opcodes, including *butterlp*, *pareq*, *hilbert*, and many others). Underflows can be avoided by:

- Using the *denorm* opcode on *ain* before *spat3d*.
- mixing low level DC or noise to the input signal, e.g.

```
atmp rnd31 1/1e24, 0, 0
```

```
aW, aX, aY, aZ spa3di ain + atmp, ...
```

or

```
aW, aX, aY, aZ spa3di ain + 1/1e24, ...
```

- reducing *irlen* in the case of *spat3dt* (which does not have an input signal). A value of about 0.005 is suitable for most uses, although it also depends on EQ settings. If the equalizer is not used, « *irlen* » can be set to 0.

## Examples

Here is a example of the *spat3d* opcode that outputs a stereo file. It uses the file *spat3d\_stereo.csd* [examples/spat3d\_stereo.csd].

### Exemple 499. Stereo example of the *spat3d* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_stereo.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr      = 48000
kr      = 1000
ksmps   = 48
nchnls  = 2

/* room parameters */

idep     = 3      /* early reflection depth      */

itmp      ftgen  1, 0, 64, -2,
              /* depth1, depth2, max delay, IR length, idist, seed */ \
              idep, 48, -1, 0.01, 0.25, 123, \
              1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceiling */ \
              1, 1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
              1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
              1, 9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back */ \
              1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
              1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2 /* left */

instr 1

/* some source signal */

a1      phasor 150              ; oscillator
a1      butterbp a1, 500, 200   ; filter
a1      = taninv(a1 * 100)
a2      phasor 3                ; envelope
a2      mirror 40*a2, -100, 5
a2      limit a2, 0, 1
a1      = a1 * a2 * 9000

kazim   line 0, 2.5, 360        ; move sound source around
kdist   line 1, 10, 4           ; distance

; convert polar coordinates
kX      = sin(kazim * 3.14159 / 180) * kdist
kY      = cos(kazim * 3.14159 / 180) * kdist
kZ      = 0

a1      = a1 + 0.000001 * 0.000001 ; avoid underflows

imode   = 1 ; change this to 3 for 8 spk in a cube,
           ; or 1 for simple stereo

aW, aX, aY, aZ spat3d a1, kX, kY, kZ, 1.0, 1, imode, 2, 2

aW      = aW * 1.4142

; stereo
;
aL      = aW + aY              /* left */
aR      = aW - aY              /* right */

; quad (square)
;
;aFL    = aW + aX + aY         /* front left */
;aFR    = aW + aX - aY         /* front right */
;aRL    = aW - aX + aY         /* rear left */
;aRR    = aW - aX - aY         /* rear right */

; eight channels (cube)
;
;aUFL   = aW + aX + aY + aZ    /* upper front left */
;aUFR   = aW + aX - aY + aZ    /* upper front right */
;aURL   = aW - aX + aY + aZ    /* upper rear left */
;aURR   = aW - aX - aY + aZ    /* upper rear right */
;aLFL   = aW + aX + aY - aZ    /* lower front left */
;aLFR   = aW + aX - aY - aZ    /* lower front right */
;aLRL   = aW - aX + aY - aZ    /* lower rear left */
;aLRR   = aW - aX - aY - aZ    /* lower rear right */

outs aL, aR

endin

</CsInstruments>

```

```
<CsScore>

/* Written by Istvan Varga */
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

Here is a example of the spat3d opcode that outputs a UHJ file. It uses the file *spat3d\_UHJ.csd* [examples/spat3d\_UHJ.csd].

### Exemple 500. UHJ example of the spat3d opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_UHJ.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr = 48000
kr = 750
ksmps = 64
nchnls = 2

itmp      ftgen      1, 0, 64, -2,
/* depth1, depth2, max delay, IR length, idist, seed */ \
3, 48, -1, 0.01, 0.25, 123, \
1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceil */ \
1, 1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
1, 9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back */ \
1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2 /* left */ \

instr 1

p3 = p3 + 1.0

kazim line 0.0, 4.0, 360.0      ; azimuth
kelev line 40, p3 - 1.0, -20   ; elevation
kdist = 2.0                    ; distance
; convert coordinates
kX = kdist * cos(kelev * 0.01745329) * sin(kazim * 0.01745329)
kY = kdist * cos(kelev * 0.01745329) * cos(kazim * 0.01745329)
kZ = kdist * sin(kelev * 0.01745329)

; source signal
a1 phasor 160.0
a2 delayl a1
a1 = a1 - a2
kffrq1 port 200.0, 0.8, 12000.0
affrq upsamp kffrq1
affrq pareq affrq, 5.0, 0.0, 1.0, 2
kffrq downsamp affrq
aenv4 phasor 3.0
aenv4 limit 2.0 - aenv4 * 8.0, 0.0, 1.0
a1 butterbp a1 * aenv4, kffrq, 160.0
aenv linseg 1.0, p3 - 1.0, 1.0, 0.04, 0.0, 1.0, 0.0
a_ = 4000000 * a1 * aenv + 0.00000001

; spatialize
a_W, a_X, a_Y, a_Z spat3d a_, kX, kY, kZ, 1.0, 1, 2, 2.0, 2

; convert to UHJ format (stereo)
aWre, aWim hilbert a_W
aXre, aXim hilbert a_X
aYre, aYim hilbert a_Y
```

```

aWXre = 0.0928*aXre + 0.4699*aWre
aWXim = 0.2550*aXim - 0.1710*aWim

aL = aWXre + aWXim + 0.3277*aYre
aR = aWXre - aWXim - 0.3277*aYre

```

```

    outs aL, aR

```

```

endin

```

```

</CsInstruments>
<CsScore>

```

```

/* Written by Istvan Varga */
t 0 60

```

```

i 1 0.0 8.0
e

```

```

</CsScore>
</CsoundSynthesizer>

```

Here is a example of the spat3d opcode that outputs a quadrophonic file. It uses the file *spat3d\_quad.csd* [examples/spat3d\_quad.csd].

### Exemple 501. Quadrophonic example of the spat3d opcode.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_quad.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr      = 48000
kr      = 1000
ksmps   = 48
nchnls  = 4

/* room parameters */

idep     = 3      /* early reflection depth      */

itmp     ftgen    1, 0, 64, -2,
/* depth1, depth2, max delay, IR length, idist, seed */ \
idep, 48, -1, 0.01, 0.25, 123, \
1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceil */ \
1, 1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
1, 9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back */ \
1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2 /* left */

instr 1

/* some source signal */

a1      phasor 150                ; oscillator
a1      butterbp a1, 500, 200    ; filter
a1      = taninv(a1 * 100)
a2      phasor 3                  ; envelope
a2      mirror 40*a2, -100, 5
a2      limit a2, 0, 1
a1      = a1 * a2 * 9000

kazim    line 0, 2.5, 360          ; move sound source around
kdist    line 1, 10, 4             ; distance

```

```

; convert polar coordinates
kX      = sin(kazim * 3.14159 / 180) * kdist
kY      = cos(kazim * 3.14159 / 180) * kdist
kZ      = 0

a1      = a1 + 0.000001 * 0.000001      ; avoid underflows

imode   = 2      ; change this to 3 for 8 spk in a cube,
                  ; or 1 for simple stereo

aW, aX, aY, aZ  spat3d a1, kX, kY, kZ, 1.0, 1, imode, 2, 2

aW      = aW * 1.4142

; stereo
;
;aL      = aW + aY      /* left          */
;aR      = aW - aY      /* right         */

; quad (square)
;
;aFL     = aW + aX + aY      /* front left    */
;aFR     = aW + aX - aY      /* front right   */
;aRL     = aW - aX + aY      /* rear left     */
;aRR     = aW - aX - aY      /* rear right    */

; eight channels (cube)
;
;aUFL    = aW + aX + aY + aZ /* upper front left */
;aUFR    = aW + aX - aY + aZ /* upper front right */
;aURL    = aW - aX + aY + aZ /* upper rear left  */
;aURR    = aW - aX - aY + aZ /* upper rear right */
;aLFL    = aW + aX + aY - aZ /* lower front left  */
;aLFR    = aW + aX - aY - aZ /* lower front right */
;aLRL    = aW - aX + aY - aZ /* lower rear left   */
;aLRR    = aW - aX - aY - aZ /* lower rear right  */

outq aFL, aFR, aRL, aRR

endin

</CsInstruments>
<CsScore>

/* Written by Istvan Varga */
t 0 60
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

## See Also

*spat3di*, *spat3dt*

## Credits

Author: Istvan Varga  
2001

New in version 4.12

Updated April 2002 by Istvan Varga

# spat3di

spat3di — Positions the input sound in a 3D space with the sound source position set at i-time.

## Description

This opcode positions the input sound in a 3D space, with optional simulation of room acoustics, in various output formats. With *spat3di*, sound source position is set at i-time.

## Syntax

```
aW, aX, aY, aZ spat3di ain, iX, iY, iZ, idist, ift, imode [, istor]
```

## Initialization

*iX* -- Sound source X coordinate in meters (positive: right, negative: left)

*iY* -- Sound source Y coordinate in meters (positive: front, negative: back)

*iZ* -- Sound source Z coordinate in meters (positive: up, negative: down)

*idist* -- For modes 0 to 3, *idist* is the unit circle distance in meters. For mode 4, *idist* is the distance between microphones.

The following formulas describe amplitude and delay as a function of sound source distance from microphone(s):

$$\text{amplitude} = 1 / (0.1 + \text{distance})$$
$$\text{delay} = \text{distance} / 340 \text{ (in seconds)}$$

Distance can be calculated as:

$$\text{distance} = \sqrt{iX^2 + iY^2 + iZ^2}$$

In Mode 4, distance can be calculated as:

$$\text{distance from left mic} = \sqrt{(iX + idist/2)^2 + iY^2 + iZ^2}$$
$$\text{distance from right mic} = \sqrt{(iX - idist/2)^2 + iY^2 + iZ^2}$$

With *spat3d* the distance between the sound source and any microphone should be at least  $(340 * 18) / \text{sr}$  meters. Shorter distances will work, but may produce artifacts in some cases. There is no such limitation for *spat3di* and *spat3dt*.

Sudden changes or discontinuities in sound source location can result in pops or clicks. Very fast movement may also degrade quality.

*ift* -- Function table storing room parameters (for free field spatialization, set it to zero or negative). Table size is 54. The values in the table are:

Room Parameter	Purpose
0	Early reflection recursion depth (0 is the sound source, 1 is the first reflection etc.) for spat3d and spat3di. The number of echoes for four walls (front, back, right, left) is: $N = (2 * R + 2) * R$ . If all six walls are enabled: $N = (((4 * R + 6) * R + 8) * R) / 3$
1	Late reflection recursion depth (used by spat3dt only). spat3dt skips early reflections and renders echoes up to this level. If early reflection depth is negative, spat3d and spat3di will output zero, while spat3dt will start rendering from the sound source.
2	imdel for spat3d. Overrides opcode parameter if non-negative.
3	irlen for spat3dt. Overrides opcode parameter if non-negative.
4	idist value. Overrides opcode parameter if $\geq 0$ .
5	Random seed (0 - 65535) -1 seeds from current time.
6 - 53	wall parameters (w = 6: ceil, w = 14: floor, w = 22: front, w = 30: back, w = 38: right, w = 46: left)
w + 0	Enable reflections from this wall (0: no, 1: yes)
w + 1	Wall distance from listener (in meters)
w + 2	Randomization of wall distance (0 - 1) (in units of $1 / (\text{wall distance})$ )
w + 3	Reflection level (-1 - 1)
w + 4	Parametric equalizer frequency in Hz.
w + 5	Parametric equalizer level (1.0: no filtering)
w + 6	Parametric equalizer Q (0.7071: no resonance)
w + 7	Parametric equalizer mode (0: peak EQ, 1: low shelf, 2: high shelf)

*imode* -- Output mode

- 0: B format with W output only (mono)

aout = aW

- 1: B format with W and Y output (stereo)



$a_{left} = aW + 0.7071 * aY$   
 $a_{right} = aW - 0.7071 * aY$

- 2: B format with W, X, and Y output (2D). This can be converted to UHJ:

$aW_{re}, aW_{im} \quad \text{hilbert } aW$   
 $aX_{re}, aX_{im} \quad \text{hilbert } aX$   
 $aY_{re}, aY_{im} \quad \text{hilbert } aY$   
 $aWX_r = 0.0928 * aX_{re} + 0.4699 * aW_{re}$   
 $aWX_iY_r = 0.2550 * aX_{im} - 0.1710 * aW_{im} + 0.3277 * aY_{re}$   
 $a_{left} = aWX_r + aWX_iY_r$   
 $a_{right} = aWX_r - aWX_iY_r$

- 3: B format with all outputs (3D)
- 4: Simulates a pair of microphones (stereo output)

$aW \quad \text{butterlp } aW, \text{ ifreq} \quad ; \text{ recommended values for ifreq}$   
 $aY \quad \text{butterlp } aY, \text{ ifreq} \quad ; \text{ are around 1000 Hz}$   
 $a_{left} = aW + aX$   
 $a_{right} = aY + aZ$

Mode 0 is the cheapest to calculate, while mode 4 is the most expensive.

In Mode 4, The optional lowpass filters can change the frequency response depending on direction. For example, if the sound source is located left to the listener then the high frequencies are attenuated in the right channel and slightly increased in the left. This effect can be disabled by not using filters. You can also experiment with other filters (tone etc.) for better effect.

Note that mode 4 is most useful for listening with headphones, and is also more expensive to calculate than the B-format (0 to 3) modes. The *idist* parameter in this case sets the distance between left and right microphone; for headphones, values between 0.2 - 0.25 are recommended, although higher settings up to 0.4 may be used for wide stereo effects.

More information about B format can be found here:  
[http://www.york.ac.uk/inst/mustech/3d\\_audio/ambis2.htm](http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm)

*istor* (optional, default=0) -- Skip initialization if non-zero (default: 0).

## Performance

*ain* -- Input signal

*aW, aX, aY, aZ* -- Output signals

	mode 0	mode 1	mode 2	mode 3	mode 4
aW	W out	W out	W out	W out	left chn / low freq.
aX	0	0	X out	X out	left chn / high freq.

	mode 0	mode 1	mode 2	mode 3	mode 4
aY	0	Y out	Y out	Y out	right chn / low frq.
aZ	0	0	0	Z out	right chn / high fr.

If you encounter very slow performance (up to 100 times slower), it may be caused by denormals (this is also true of many other IIR opcodes, including *butterlp*, *pareq*, *hilbert*, and many others). Underflows can be avoided by:

- Using the *denorm* opcode on *ain* before *spat3di*.
- mixing low level DC or noise to the input signal, e.g.

atmp rnd31 1/1e24, 0, 0

aW, aX, aY, aZ spat3di ain + atmp, ...

or

aW, aX, aY, aZ spa3di ain + 1/1e24, ...

- reducing *irlen* in the case of *spat3dt* (which does not have an input signal). A value of about 0.005 is suitable for most uses, although it also depends on EQ settings. If the equalizer is not used, « *irlen* » can be set to 0.

## Examples

See the examples for *spat3d*.

## See Also

*spat3d*, *spat3dt*

## Credits

Author: Istvan Varga  
2001

New in version 4.12

Updated April 2002 by Istvan Varga

# spat3dt

spat3dt — Can be used to render an impulse response for a 3D space at i-time.

## Description

This opcode positions the input sound in a 3D space, with optional simulation of room acoustics, in various output formats. *spat3dt* can be used to render the impulse response at i-time, storing output in a function table, suitable for convolution.

## Syntax

```
spat3dt ioutft, iX, iY, iZ, idist, ift, imode, irlen [, iftnocl]
```

## Initialization

*ioutft* -- Output ftable number for spat3dt. W, X, Y, and Z outputs are written interleaved to this table. If the table is too short, output will be truncated.

*iX* -- Sound source X coordinate in meters (positive: right, negative: left)

*iY* -- Sound source Y coordinate in meters (positive: front, negative: back)

*iZ* -- Sound source Z coordinate in meters (positive: up, negative: down)

*idist* -- For modes 0 to 3, *idist* is the unit circle distance in meters. For mode 4, *idist* is the distance between microphones.

The following formulas describe amplitude and delay as a function of sound source distance from microphone(s):

$$\text{amplitude} = 1 / (0.1 + \text{distance})$$

$$\text{delay} = \text{distance} / 340 \text{ (in seconds)}$$

Distance can be calculated as:

$$\text{distance} = \sqrt{iX^2 + iY^2 + iZ^2}$$

In Mode 4, distance can be calculated as:

$$\text{distance from left mic} = \sqrt{(iX + idist/2)^2 + iY^2 + iZ^2}$$

$$\text{distance from right mic} = \sqrt{(iX - idist/2)^2 + iY^2 + iZ^2}$$

With *spat3d* the distance between the sound source and any microphone should be at least  $(340 * 18) / \text{sr}$  meters. Shorter distances will work, but may produce artifacts in some cases. There is no such limitation for *spat3di* and *spat3dt*.

Sudden changes or discontinuities in sound source location can result in pops or clicks. Very fast movement may also degrade quality.

*ift* -- Function table storing room parameters (for free field spatialization, set it to zero or negative). Table size is 54. The values in the table are:

Room Parameter	Purpose
0	Early reflection recursion depth (0 is the sound source, 1 is the first reflection etc.) for spat3d and spat3di. The number of echoes for four walls (front, back, right, left) is: $N = (2 * R + 2) * R$ . If all six walls are enabled: $N = (((4 * R + 6) * R + 8) * R) / 3$
1	Late reflection recursion depth (used by spat3dt only). spat3dt skips early reflections and renders echoes up to this level. If early reflection depth is negative, spat3d and spat3di will output zero, while spat3dt will start rendering from the sound source.
2	imdel for spat3d. Overrides opcode parameter if non-negative.
3	irlen for spat3dt. Overrides opcode parameter if non-negative.
4	idist value. Overrides opcode parameter if $\geq 0$ .
5	Random seed (0 - 65535) -1 seeds from current time.
6 - 53	wall parameters (w = 6: ceil, w = 14: floor, w = 22: front, w = 30: back, w = 38: right, w = 46: left)
w + 0	Enable reflections from this wall (0: no, 1: yes)
w + 1	Wall distance from listener (in meters)
w + 2	Randomization of wall distance (0 - 1) (in units of $1 / (\text{wall distance})$ )
w + 3	Reflection level (-1 - 1)
w + 4	Parametric equalizer frequency in Hz.
w + 5	Parametric equalizer level (1.0: no filtering)
w + 6	Parametric equalizer Q (0.7071: no resonance)
w + 7	Parametric equalizer mode (0: peak EQ, 1: low shelf, 2: high shelf)

*imode* -- Output mode

- 0: B format with W output only (mono)

aout = aW

- 1: B format with W and Y output (stereo)

```
aleft = aW + 0.7071*aY
aright = aW - 0.7071*aY
```

- 2: B format with W, X, and Y output (2D). This can be converted to UHJ:

```
aWre, aWim    hilbert aW
aXre, aXim    hilbert aX
aYre, aYim    hilbert aY
aWXr = 0.0928*aXre + 0.4699*aWre
aWXiYr = 0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aleft = aWXr + aWXiYr
aright = aWXr - aWXiYr
```

- 3: B format with all outputs (3D)
- 4: Simulates a pair of microphones (stereo output)

```
aW    butterlp aW, ifreq    ; recommended values for ifreq
aY    butterlp aY, ifreq    ; are around 1000 Hz
aleft = aW + aX
aright = aY + aZ
```

Mode 0 is the cheapest to calculate, while mode 4 is the most expensive.

In Mode 4, The optional lowpass filters can change the frequency response depending on direction. For example, if the sound source is located left to the listener then the high frequencies are attenuated in the right channel and slightly increased in the left. This effect can be disabled by not using filters. You can also experiment with other filters (tone etc.) for better effect.

Note that mode 4 is most useful for listening with headphones, and is also more expensive to calculate than the B-format (0 to 3) modes. The *idist* parameter in this case sets the distance between left and right microphone; for headphones, values between 0.2 - 0.25 are recommended, although higher settings up to 0.4 may be used for wide stereo effects.

More information about B format can be found here:  
[http://www.york.ac.uk/inst/mustech/3d\\_audio/ambis2.htm](http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm)

*irlen* -- Impulse response length of echoes (in seconds). Depending on filter parameters, values around 0.005-0.01 are suitable for most uses (higher values result in more accurate output, but slower rendering)

*iftnocl* (optional, default=0) -- Do not clear output ftable (mix to existing data) if set to 1, clear table before writing if set to 0 (default: 0).

## Examples

See the examples for *spat3d*.

## See Also

*spat3d, spat3di*

## Credits

Author: Istvan Varga  
2001

New in version 4.12

Updated April 2002 by Istvan Varga

# spdist

spdist — Calculates distance values from xy coordinates.

## Description

*spdist* uses the same xy data as *space*, also either from a text file using *Gen28* or from x and y arguments given to the unit directly. The purpose of this unit is to make available the values for distance that are calculated from the xy coordinates.

In the case of *space*, the xy values are used to determine a distance which is used to attenuate the signal and prepare it for use in *spsend*. But it is also useful to have these values for distance available to scale the frequency of the signal before it is sent to the *space* unit.

## Syntax

```
k1 spdist ifn, ktime, kx, ky
```

## Initialization

*ifn* -- number of the stored function created using *Gen28*. This function generator reads a text file which contains sets of three values representing the xy coordinates and a time-tag for when the signal should be placed at that location. The file should look like:

```
0   -1   1
1    1   1
2    4   4
2.1 -4  -4
3   10 -10
5  -40   0
```

If that file were named "move" then the *Gen28* call in the score would like:

```
f1 0 0 28 "move"
```

*Gen28* takes 0 as the size and automatically allocates memory. It creates values to 10 milliseconds of resolution. So in this case there will be 500 values created by interpolating X1 to X2 to X3 and so on, and Y1 to Y2 to Y3 and so on, over the appropriate number of values that are stored in the function table. In the above example, the sound will begin in the left front, over 1 second it will move to the right front, over another second it move further into the distance but still in the left front, then in just 1/10th of a second it moves to the left rear, a bit distant. Finally over the last .9 seconds the sound will move to the right rear, moderately distant, and it comes to rest between the two left channels (due west!), quite distant. Since the values in the table are accessed through the use of a time-pointer in the *space* unit, the actual timing can be made to follow the file's timing exactly or it can be made to go faster or slower through the same trajectory. If you have access to the GUI that allows one to draw and edit the files, there is no need to create the text files manually. But as long as the file is ASCII and in the format shown above, it doesn't matter how it is made!

IMPORTANT: If *ifn* is 0 then *space* will take its values for the xy coordinates from *kx* and *ky*.

## Performance

The configuration of the xy coordinates in space places the signal in the following way:

- a1 is -1, 1
- a2 is 1, 1
- a3 is -1, -1
- a4 is 1, -1

This assumes a loudspeaker set up as a1 is left front, a2 is right front, a3 is left back, a4 is right back. Values greater than 1 will result in sounds being attenuated, as if in the distance. *space* considers the speakers to be at a distance of 1; smaller values of xy can be used, but *space* will not amplify the signal in this case. It will, however balance the signal so that it can sound as if it were within the 4 speaker *space*. x=0, y=1, will place the signal equally balanced between left and right front channels, x=y=0 will place the signal equally in all 4 channels, and so on. Although there must be 4 output signals from *space*, it can be used in a 2 channel orchestra. If the xy's are kept so that  $Y \geq 1$ , it should work well to do panning and fixed localization in a stereo field.

*ktime* -- index into the table containing the xy coordinates. If used like:

```
ktime      line 0, 5, 5
a1, a2, a3, a4 space asig, 1, ktime, ...
```

with the file "move" described above, the speed of the signal's movement will be exactly as described in that file. However:

```
ktime      line 0, 10, 5
```

the signal will move at half the speed specified. Or in the case of:

```
ktime      line 5, 15, 0
```

the signal will move in the reverse direction as specified and 3 times slower! Finally:

```
ktime      line 2, 10, 3
```

will cause the signal to move only from the place specified in line 3 of the text file to the place specified in line 5 of the text file, and it will take 10 seconds to do it.

*kx*, *ky* -- when *ifn* is 0, *space* and *spdist* will use these values as the XY coordinates to localize the signal.



## Examples

```
instr 1
  asig      ;some audio signal
  ktime          line 0, p3, p10
  a1, a2, a3, a4      space asig,1, ktime, .1
  ar1, ar2, ar3, ar4  spsend

  ga1 = ga1+ar1
  ga2 = ga2+ar2
  ga3 = ga3+ar3
  ga4 = ga4+ar4

                                outq a1, a2, a3, a4
endin

instr 99 ; reverb instrument

  a1 reverb2 ga1, 2.5, .5
  a2 reverb2 ga2, 2.5, .5
  a3 reverb2 ga3, 2.5, .5
  a4 reverb2 ga4, 2.5, .5

  outq a1, a2, a3, a4
  ga1=0
  ga2=0
  ga3=0
  ga4=0
```

In the above example, the signal, *asig*, is moved according to the data in Function #1 indexed by *ktime*. *space* sends the appropriate amount of the signal internally to *spsend*. The outputs of the *spsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

*space* can be useful for quad and stereo panning as well as fixed placement of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field using xy values from the score instead of a function table.

```
instr 1
  ...
  a1, a2, a3, a4      space asig, 0, 0, .1, p4, p5
  ar1, ar2, ar3, ar4  spsend

  ga1=ga1+ar1
  ga2=ga2+ar2

                                outs  a1, a2
endin

instr 99 ; reverb...
  ....
endin
```

A few notes: p4 and p5 are the X and Y values

```
;place the sound in the left speaker and near
il 0 1 -1 1
;place the sound in the right speaker and far
il 1 1 45 45
;place the sound equally between left and right and in the middle ground distance
il 2 1 0 12
e
```

The next example shows a simple intuitive use of the distance values returned by *spdist* to simulate Doppler shift.

```
ktime          line 0, p3, 10
kdist          spdist 1, ktime
kfreq = (ifreq * 340) / (340 + kdist)
asig          oscili iamp, kfreq, 1

a1, a2, a3, a4  space asig, 1, ktime, .1
ar1, ar2, ar3, ar4 spsend
```

The same function and time values are used for both *spdist* and *space*. This insures that the distance values used internally in the *space* unit will be the same as those returned by *spdist* to give the impression of a Doppler shift!

## See Also

*space, spsend*

## Credits

Author: Richard Karpen  
Seattle, WA USA  
1998

New in Csound version 3.48

# specaddm

specaddm — Perform a weighted add of two input spectra.

## Description

Perform a weighted add of two input spectra.

## Syntax

```
wsig specaddm wsig1, wsig2 [, imul2]
```

## Initialization

*imul2* (optional, default=0) -- if non-zero, scale the *wsig2* magnitudes before adding. The default value is 0.

## Performance

*wsig1* -- the first input spectra.

*wsig2* -- the second input spectra.

Do a weighted add of two input spectra. For each channel of the two input spectra, the two magnitudes are combined and written to the output according to:

$$\text{magout} = \text{mag1in} + \text{mag2in} * \text{imul2}$$

The operation is performed whenever the input *wsig1* is sensed to be new. This unit will (at Initialization) verify the consistency of the two spectra (equal size, equal period, equal mag types).

## Examples

```
wsig2  specdiff      wsig1      ; sense onsets
wsig3  specfilt      wsig2, 2    ; absorb slowly
       specdisp      wsig2, .1   ; & display both spectra
       specdisp      wsig3, .1
```

## See Also

*specdiff*, *specfilt*, *spechist*, *specscal*

# specdiff

specdiff — Finds the positive difference values between consecutive spectral frames.

## Description

Finds the positive difference values between consecutive spectral frames.

## Syntax

```
wsig specdiff wsignin
```

## Performance

*wsig* -- the output spectrum.

*wsignin* -- the input spectra.

Finds the positive difference values between consecutive spectral frames. At each new frame of *wsignin*, each magnitude value is compared with its predecessor, and the positive changes written to the output spectrum. This unit is useful as an energy onset detector.

## Examples

```
wsig2  specdiff      wsig1      ; sense onsets
wsig3  specfilt      wsig2, 2    ; absorb slowly
        specdisp      wsig2, .1  ; & display both spectra
        specdisp      wsig3, .1
```

## See Also

*specaddm*, *specfilt*, *spechist*, *specscal*

# specdisp

specdisp — Displays the magnitude values of the spectrum.

## Description

Displays the magnitude values of the spectrum.

## Syntax

```
specdisp wsig, iprd [, iwtflg]
```

## Initialization

*iprd* -- the period, in seconds, of each new display.

*iwtflg* (optional, default=0) -- wait flag. If non-zero, hold each display until released by the user. The default value is 0 (no wait).

## Performance

*wsig* -- the input spectrum.

Displays the magnitude values of spectrum *wsig* every *iprd* seconds (rounded to some integral number of *wsig*'s originating *iprd*).

## Examples

```

ksum      specsum  wsig, 1      ; sum the spec bins, and ksmooth
          if      ksum < 2000  kgoto zero ; if sufficient amplitude
koct      specptrk wsig          ; pitch-track the signal
          kgoto   contin
zero:
  koct      =      0              ; else output zero
contin:
```

## See Also

*specsum*

# specfilt

specfilt — Filters each channel of an input spectrum.

## Description

Filters each channel of an input spectrum.

## Syntax

```
wsig specfilt wsigin, ifhtim
```

## Initialization

*ifhtim* -- half-time constant.

## Performance

*wsigin* -- the input spectrum.

Filters each channel of an input spectrum. At each new frame of *wsigin*, each magnitude value is injected into a 1st-order lowpass recursive filter, whose half-time constant has been initially set by sampling the ftable *ifhtim* across the (logarithmic) frequency space of the input spectrum. This unit effectively applies a *persistence* factor to the data occurring in each spectral channel, and is useful for simulating the *energy integration* that occurs during auditory perception. It may also be used as a time-attenuated running *histogram* of the spectral distribution.

## Examples

```
wsig2  specdiff      wsig1      ; sense onsets
wsig3  specfilt      wsig2, 2    ; absorb slowly
       specdisp      wsig2, .1   ; & display both spectra
       specdisp      wsig3, .1
```

## See Also

*specaddm, specdiff, spechist, specscal*

# spechist

spechist — Accumulates the values of successive spectral frames.

## Description

Accumulates the values of successive spectral frames.

## Syntax

`wsig spechist wsign`

## Performance

*wsign* -- the input spectra.

Accumulates the values of successive spectral frames. At each new frame of *wsign*, the accumulations-to-date in each magnitude track are written to the output spectrum. This unit thus provides a running *histogram* of spectral distribution.

## Examples

```
wsig2    specdiff      wsig1      ; sense onsets
wsig3    specfilt      wsig2, 2    ; absorb slowly
          specdisp      wsig2, .1   ; & display both spectra
          specdisp      wsig3, .1
```

## See Also

*specaddm, specdiff, specfilt, specscal*

# specptrk

specptrk — Estimates the pitch of the most prominent complex tone in the spectrum.

## Description

Estimate the pitch of the most prominent complex tone in the spectrum.

## Syntax

```
koct, kamp specptrk wsig, kvar, ilo, ihi, istr, idbthresh, inptls, \  
  irolloff [, iodd] [, iconfs] [, interp] [, ifprd] [, iwtflg]
```

## Initialization

*ilo, ihi, istr* -- pitch range conditioners (low, high, and starting) expressed in decimal octave form.

*idbthresh* -- energy threshold (in decibels) for pitch tracking to occur. Once begun, tracking will be continuous until the energy falls below one half the threshold (6 dB down), whence the *koct* and *kamp* outputs will be zero until the full threshold is again surpassed. *idbthresh* is a guiding value. At initialization it is first converted to the *idbout* mode of the source spectrum (and the 6 dB down point becomes .5, .25, or 1/root 2 for modes 0, 2 and 3). The values are also further scaled to allow for the weighted partial summation used during correlation. The actual thresholding is done using the internal weighted and summed *kamp* value that is visible as the second output parameter.

*inptls, irolloff* -- number of harmonic partials used as a matching template in the spectrally-based pitch detection, and an amplitude rolloff for the set expressed as some fraction per octave (linear, so don't roll off to negative). Since the partials and rolloff fraction can affect the pitch following, some experimentation will be useful: try 4 or 5 partials with .6 rolloff as an initial setting; raise to 10 or 12 partials with rolloff .75 for complex timbres like the bassoon (weak fundamental). Computation time is dependent on the number of partials sought. The maximum number is 16.

*iodd* (optional) -- if non-zero, employ only odd partials in the above set (e.g. *inptls* of 4 would employ partials 1,3,5,7). This improves the tracking of some instruments like the clarinet. The default value is 0 (employ all partials).

*iconfs* (optional) -- number of confirmations required for the pitch tracker to jump an octave, pro-rated for fractions of an octave (i.e. the value 12 implies a semitone change needs 1 confirmation (two hits) at the *spectrum* generating *iprd*). This parameter limits spurious pitch analyses such as octave errors. A value of 0 means no confirmations required; the default value is 10.

*interp* (optional) -- if non-zero, interpolate each output signal (*koct*, *kamp*) between incoming *wsig* frames. The default value is 0 (repeat the signal values between frames).

*ifprd* (optional) -- if non-zero, display the internally computed spectrum of candidate fundamentals. The default value is 0 (no display).

*iwtflg* (optional) -- wait flag. If non-zero, hold each display until released by the user. The default value is 0 (no wait).

## Performance

At note initialization this unit creates a template of *inptls* harmonically related partials (odd partials, if



*iodd* non-zero) with amplitude rolloff to the fraction *iroloff* per octave. At each new frame of *wsig*, the spectrum is cross-correlated with this template to provide an internal spectrum of candidate fundamentals (optionally displayed). A likely pitch/amp pair (*koct*, *kamp*, in decimal octave and summed *idbout* form) is then estimated. *koct* varies from the previous *koct* by no more than plus or minus *kvar* decimal octave units. It is also guaranteed to lie within the hard limit range *ilo* -- *ihi* (decimal octave low and high pitch). *kvar* can be dynamic, e.g. onset amp dependent. Pitch resolution uses the originating *spectrum ifrqs* bins/octave, with further parabolic interpolation between adjacent bins. Settings of root magnitude, *ifrqs* = 24, *iq* = 15 should capture all the inflections of interest. Between frames, the output is either repeated or interpolated at the k-rate. (See *spectrum*.)

## Examples

```

a1,a2  ins                                ; read a stereo clarinet input
krms    rms                               ; find a monaural rms value
kvar    = 0.6 + krms/8000                 ; & use to gate the pitch varianc
wsig    spectrum                          ; get a 7-oct spectrum, 24 bibs/c
        specdisp                          ; display this and now estimate
koct,ka spectrk                          ; the pch and amp
aosc    oscil                             ; & generate \ new tone with thes
koct    = (koct<7.0?7.0:koct),2           ; replace non pitch with low C
        display                          ; & display the pitch track
        display                          ; plus the summed root mag
        outs                             ; output 1 original and 1 new tra
        al, aosc

```

# specscal

specscal — Scales an input spectral datablock with spectral envelopes.

## Description

Scales an input spectral datablock with spectral envelopes.

## Syntax

```
wsig specscal wsignin, ifscale, ifthresh
```

## Initialization

*ifscale* -- scale function table. A function table containing values by which a value's magnitude is rescaled.

*ifthresh* -- threshold function table. If *ifthresh* is non-zero, each magnitude is reduced by its corresponding table-value (to not less than zero)

## Performance

*wsig* -- the output spectrum

*wsignin* -- the input spectra

Scales an input spectral datablock with spectral envelopes. Function tables *ifthresh* and *ifscale* are initially sampled across the (logarithmic) frequency space of the input spectrum; then each time a new input spectrum is sensed the sampled values are used to scale each of its magnitude channels as follows: if *ifthresh* is non-zero, each magnitude is reduced by its corresponding table-value (to not less than zero); then each magnitude is rescaled by the corresponding *ifscale* value, and the resulting spectrum written to *wsig*.

## Examples

```
wsig2    specdiff      wsig1          ; sense onsets
wsig3    specfilt      wsig2, 2        ; absorb slowly
          specdisp      wsig2, .1      ; & display both spectra
          specdisp      wsig3, .1
```

## See Also

*specaddm*, *specdiff*, *specfilt*, *spechist*

# specsum

specsum — Sums the magnitudes across all channels of the spectrum.

## Description

Sums the magnitudes across all channels of the spectrum.

## Syntax

ksum **specsum** wsig [, interp]

## Initialization

*interp* (optional, default-0) -- if non-zero, interpolate the output signal (*koct* or *ksum*). The default value is 0 (repeat the signal value between changes).

## Performance

*ksum* -- the output signal.

*wsig* -- the input spectrum.

Sums the magnitudes across all channels of the spectrum. At each new frame of *wsig*, the magnitudes are summed and released as a scalar *ksum* signal. Between frames, the output is either repeated or interpolated at the k-rate. This unit produces a k-signal summation of the magnitudes present in the spectral data, and is thereby a running measure of its moment-to-moment overall strength.

## Examples

```

ksum      specsum  wsig, 1      ; sum the spec bins, and ksmooth
          if      ksum < 2000  kgoto zero ; if sufficient amplitude
koc      specptrk  wsig          ; pitch-track the signal
          kgoto   contin
zero:
  koc      =      0              ; else output zero
contin:

```

## See Also

*specdisp*

# spectrum

spectrum — Generate a constant-Q, exponentially-spaced DFT.

## Description

Generate a constant-Q, exponentially-spaced DFT across all octaves of a multiply-downsampled control or audio input signal.

## Syntax

```
wsig spectrum xsig, iprd, iocts, ifrqa [, iq] [, ihann] [, idbout] \  
      [, idsprd] [, idsinrs]
```

## Initialization

*ihann* (optional) -- apply a Hamming or Hanning window to the input. The default is 0 (Hamming window)

*idbout* (optional) -- coded conversion of the DFT output:

- 0 = magnitude
- 1 = dB
- 2 = mag squared
- 3 = root magnitude

The default value is 0 (magnitude).

*idsprd* (optional) -- if non-zero, display the composite downsampling buffer every *idsprd* seconds. The default value is 0 (no display).

*idsins* (optional) -- if non-zero, display the Hamming or Hanning windowed sinusoids used in DFT filtering. The default value is 0 (no sinusoid display).

## Performance

This unit first puts signal *asig* or *ksig* through *iocts* of successive octave decimation and downsampling, and preserves a buffer of down-sampled values in each octave (optionally displayed as a composite buffer every *idsprd* seconds). Then at every *iprd* seconds, the preserved samples are passed through a filter bank (*ifrqs* parallel filters per octave, exponentially spaced, with frequency/bandwidth Q of *iq*), and the output magnitudes optionally converted (*idbout*) to produce a band-limited spectrum that can be read by other units.

The stages in this process are computationally intensive, and computation time varies directly with *iocts*, *ifrqs*, *iq*, and inversely with *iprd*. Settings of *ifrqs* = 12, *iq* = 10, *idbout* = 3, and *iprd* = .02 will normally be adequate, but experimentation is encouraged. *ifrqs* currently has a maximum of 120 divisions per octave. For audio input, the frequency bins are tuned to coincide with A440.

This unit produces a self-defining spectral datablock *wsig*, whose characteristics used (*iprd*, *iocts*, *ifrqs*, *idbout*) are passed via the data block itself to all derivative *wsigs*. There can be any number of spectrum

units in an instrument or orchestra, but all *wsig* names must be unique.

## Examples

```
asig in                                ; get external audio
wsig spectrum asig,.02,6,12,33,0,1,1 ; downsample in 6 octs & calc a 72 pt dft (Q 33, dB out) every 2
```

# splitrig

splitrig — Split a trigger signal

## Description

*splitrig* splits a trigger signal (i.e. a timed sequence of control-rate impulses) into several channels following a structure designed by the user.

## Syntax

```
splitrig ktrig, kndx, imaxtics, ifn, kout1 [,kout2,...,koutN]
```

## Initialization

*imaxtics* - number of tics belonging to largest pattern

*ifn* - number of table containing channel-data structuring

## Performance

*asig* - incoming (input) signal

*ktrig* - trigger signal

The *splitrig* opcode splits a trigger signal into several output channels according to one or more patterns provided by the user. Normally the regular timed trigger signal generated by metro opcode is used to be transformed into rhythmic pattern that can trig several independent melodies or percussion riffs. But you can also start from non-isocronous trigger signals. This allows to use some "interpretative" and less "mechanic" groove variations. Patterns are looped and each numtics\_of\_pattern\_N the cycle is repeated.

The scheme of patterns is defined by the user and is stored into ifn table according to the following format:

```
gil ftgen 1,0,1024, -2 \ ; table is generated with GEN02 in this case
\
numtics_of_pattern_1, \ ;pattern 1
    tic1_out1, tic1_out2, ... , tic1_outN,\
    tic2_out1, tic2_out2, ... , tic2_outN,\
    tic3_out1, tic3_out2, ... , tic3_outN,\
    .....
    ticN_out1, ticN_out2, ... , ticN_outN,\
\
numtics_of_pattern_2, \ ;pattern 2
    tic1_out1, tic1_out2, ... , tic1_outN,\
    tic2_out1, tic2_out2, ... , tic2_outN,\
    tic3_out1, tic3_out2, ... , tic3_outN,\
    .....
    ticN_out1, ticN_out2, ... , ticN_outN,\
    .....
\
numtics_of_pattern_N, \ ;pattern N
    tic1_out1, tic1_out2, ... , tic1_outN,\
    tic2_out1, tic2_out2, ... , tic2_outN,\
    tic3_out1, tic3_out2, ... , tic3_outN,\
    .....
    ticN_out1, ticN_out2, ... , ticN_outN,\
```

This scheme can contain more than one pattern, each one with a different number of rows. Each pattern is preceded by a special row containing a single *numtics\_of\_pattern\_N* field; this field expresses the number of tics that makes up the corresponding pattern. Each pattern's row makes up a tic. Each pattern's column corresponds to a channel, and each field of a row is a number that makes up the value outputted by the corresponding *koutXX* channel (if number is a zero, corresponding output channel will not trigger anything in that particular arguments). Obviously, all rows must contain the same number of fields that must be equal to the number of *koutXX* channel. All patterns must contain the same number of rows, this number must be equal to the largest pattern and is defined by *imaxtics* variable. Even if a pattern has less tics than the largest pattern, it must be made up of the same number of rows, in this case, some of these rows, at the end of the pattern itself, will not be used (and can be set to any value, because it doesn't matter).

The *kndx* variable chooses the number of the pattern to be played, zero indicating the first pattern. Each time the integer part of *kndx* changes, tic counter is reset to zero.

Patterns are looped and each *numtics\_of\_pattern\_N* the cycle is repeated.

examples 4 - calculate average value of *asig* in the time interval

This opcode can be useful in several situations, for example to implement a vu-meter

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# spsend

spsend — Generates output signals based on a previously defined *space* opcode.

## Description

*spsend* depends upon the existence of a previously defined *space*. The output signals from *spsend* are derived from the values given for *xy* and *reverb* in the *space* and are ready to be sent to local or global reverb units (see example below).

## Syntax

a1, a2, a3, a4 **spsend**

## Performance

The configuration of the *xy* coordinates in *space* places the signal in the following way:

- a1 is -1, 1
- a2 is 1, 1
- a3 is -1, -1
- a4 is 1, -1

This assumes a loudspeaker set up as a1 is left front, a2 is right front, a3 is left back, a4 is right back. Values greater than 1 will result in sounds being attenuated, as if in the distance. *space* considers the speakers to be at a distance of 1; smaller values of *xy* can be used, but *space* will not amplify the signal in this case. It will, however balance the signal so that it can sound as if it were within the 4 speaker *space*. *x*=0, *y*=1, will place the signal equally balanced between left and right front channels, *x*=*y*=0 will place the signal equally in all 4 channels, and so on. Although there must be 4 output signals from *space*, it can be used in a 2 channel orchestra. If the *xy*'s are kept so that *Y* >= 1, it should work well to do panning and fixed localization in a stereo field.

## Examples

```
instr 1
  asig      ;some audio signal
  ktime          line 0, p3, p10
  a1, a2, a3, a4  space asig,1, ktime, .1
  ar1, ar2, ar3, ar4 spsend

  ga1 = ga1+ar1
  ga2 = ga2+ar2
  ga3 = ga3+ar3
  ga4 = ga4+ar4

                                outq a1, a2, a3, a4
endin

instr 99 ; reverb instrument
```



```

a1 reverb2 ga1, 2.5, .5
a2 reverb2 ga2, 2.5, .5
a3 reverb2 ga3, 2.5, .5
a4 reverb2 ga4, 2.5, .5

    outq a1, a2, a3, a4
ga1=0
ga2=0
ga3=0
ga4=0

```

In the above example, the signal, *asig*, is moved according to the data in Function #1 indexed by *ktime*. *space* sends the appropriate amount of the signal internally to *spsend*. The outputs of the *spsend* are added to global accumulators in a common Csound style and the global signals are used as inputs to the reverb units in a separate instrument.

*space* can be useful for quad and stereo panning as well as fixed placement of sounds anywhere between two loudspeakers. Below is an example of the fixed placement of sounds in a stereo field using *xy* values from the score instead of a function table.

```

instr 1
...
a1, a2, a3, a4      space asig, 0, 0, .1, p4, p5
ar1, ar2, ar3, ar4 spsend

ga1=ga1+ar1
ga2=ga2+ar2

                                outs  a1, a2
endin

instr 99 ; reverb...
....
endin

```

A few notes: *p4* and *p5* are the X and Y values

```

;place the sound in the left speaker and near
i1 0 1 -1 1
;place the sound in the right speaker and far
i1 1 1 45 45
;place the sound equally between left and right and in the middle ground distance
i1 2 1 0 12
e

```

The next example shows a simple intuitive use of the distance values returned by *spdist* to simulate Doppler shift.

```

ktime          line 0, p3, 10
kdist          spdist 1, ktime
kfreq = (ifreq * 340) / (340 + kdist)
asig           oscili iamp, kfreq, 1

a1, a2, a3, a4      space asig, 1, ktime, .1
ar1, ar2, ar3, ar4 spsend

```

The same function and time values are used for both *spdist* and *space*. This insures that the distance values used internally in the *space* unit will be the same as those returned by *spdist* to give the impression of a Doppler shift!

## See Also

*space, spdist*

## Credits

Author: Richard Karpen  
Seattle, WA USA  
1998

New in Csound version 3.48

# sprintf

`sprintf` — printf-style formatted output to a string variable.

## Description

*sprintf* write printf-style formatted output to a string variable, similarly to the C function `sprintf()`. `sprintf` runs at i-time only.

## Syntax

```
Sdst sprintf Sfmt, xarg1[, xarg2[, ... ]]
```

```
Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]
```

## Initialization

*Sfmt* -- format string, has the same format as in `printf()` and other similar C functions, except length modifiers (l, ll, h, etc.) are not supported. The following conversion specifiers are allowed:

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

*xarg1*, *xarg2*, ... -- input arguments (max. 30) for format, should be i-rate for all conversion specifiers except %s, which requires a string argument. Integer formats like %d round the input values to the nearest integer.

## Performance

*Sdst* -- output string variable

## Example

```
Sname    sprintf "soundin-%04d.wav", ifileno
Smsg     sprintf "The file name is: '%s'", Sname
         puts Smsg, 1
asig soundin Sname
```

## See also

*sprintfk*

## Credits

Author: Istvan Varga  
2005

# sprintfk

sprintfk — printf-style formatted output to a string variable at k-rate.

## Description

*sprintfk* writes printf-style formatted output to a string variable, similarly to the C function `sprintf()`. *sprintfk* runs both at initialization and performance time.

## Syntax

```
Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]
```

## Initialization

*Sfmt* -- format string, has the same format as in `printf()` and other similar C functions, except length modifiers (l, ll, h, etc.) are not supported. The following conversion specifiers are allowed:

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

*xarg1*, *xarg2*, ... -- input arguments (max. 30) for format, should be i-rate for all conversion specifiers except %s, which requires a string argument. *sprintfk* also allows k-rate number arguments, but these should still be valid at init time as well (unless *sprintfk* is skipped with *igoto*). Integer formats like %d round the input values to the nearest integer.

## Performance

*Sdst* -- output string variable

## Examples

Here is an example of the *sprintfk* opcode. It uses the file *sprintfk.csd* [examples/sprintfk.csd].

### Exemple 502. Example of the *sprintfk* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sprintfk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 48000
ksmps   = 16
nchnls  = 2
0dbfs   = 1
```

```

; Example by Jonathan Murphy 2007

instr 1

S1      = "1"
S2      = " + 1"
ktrig   =   init      0
kval     =   init      2
if (ktrig == 1) then
  S1     =   strcatk   S1, S2
  kval   =   kval + 1
endif
String   =   sprintfk  "%s = %d", S1, kval
puts     =   String, kval
ktrig    =   metro     1

endin

</CsInstruments>
<CsScore>
il 0 10
e
</CsScore>
</CsoundSynthesizer>

```

## See also

*sprintf, puts, strcat*

## Credits

Author: Istvan Varga  
 2005  
 Example by Jonathan Murphy

# sqrt

sqrt — Retourne une racine carrée.

## Description

Retourne la racine carrée de  $x$  ( $x$  non-négatif).

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

## Syntaxe

**sqrt**( $x$ ) (pas de restriction de taux)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode sqrt. Il utilise le fichier *sqrt.csd* [exemples/sqrt.csd].

### Exemple 503. Exemple de l'opcode sqrt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sqrt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = sqrt(64)
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i1 = 8.000
```

## Voir Aussi

*abs, exp, frac, int, log, log10, i*

## Crédits

Exemple écrit par Kevin Conder.

# sr

sr — Fixe la taux d'échantillonnage audio.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

```
sr = iarg
```

## Initialisation

*sr* = (facultatif) -- fixe le taux d'échantillonnage à *iarg* échantillons par seconde par canal. La valeur par défaut est 44100.

De plus, toute *variable globale* [56] peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *sr*. Le taux d'échantillonnage sera calculé à partir de *kr* et de *ksmps*, mais le résultat doit être une valeur entière. Si aucune de ces valeurs globales n'est définie, le taux d'échantillonnage par défaut sera 44100. Habituellement, vous utiliserez une valeur supportée par votre carte son, comme 44100 ou 48000, sinon, le résultat audio généré par csound risque d'être injouable, ou bien vous aurez une erreur si vous essayez une exécution en temps-réel. Vous pouvez naturellement utiliser un taux d'échantillonnage comme 96000, pour un rendu différé, même si votre carte son ne le supporte pas. Csound générera un fichier valide jouable sur des systèmes offrant cette possibilité.

## Exemples

```
sr = 10000
kr = 500
ksmps = 20
gil = sr/2.
ga init 0
itranspose = octpch(.01)
```

## Voir Aussi

*kr*, *ksmps*, *nchnls*



# stack

`stack` — Initializes the stack.

## Description

Initializes and sets the size of the global stack.

## Syntax

```
stack iStackSize
```

## Initialization

*iStackSize* - size of the stack in bytes.

## Performance

Csound implements a single global stack. Initializing the stack with the *stack* opcode is not required - it is optional, and if not done, the first use of *push* or *push\_f* will automatically create a stack of 32768 bytes. Otherwise, *stack* is normally called from the orchestra header, and takes a stack size parameter in bytes (there is an upper limit of about 16 MB). Once set, the stack size is fixed and cannot be changed during performance.

The global stack works in LIFO order: after multiple *push* calls, *pop* should be used in reverse order.

Each *push* or *pop* operation can work on a "bundle" of multiple variables. When using *pop*, the number, type, and order of items must match those used by the corresponding *push*. That is, after a 'push Sfoo, ibar', you must call something like 'pop Sbar, ifoo', and not e.g. two separate 'pop' statements.

*push* and *pop* opcodes can take variables of any type (i-, k-, a- and strings). Variables of type 'a' and 'k' are passed at performance time only, while 'i' and 'S' are passed at init time only.

push/pop for a, k, i, and S types copy data by value. By contrast, *push\_f* only pushes a "reference" to the f-signal, and then the corresponding *pop\_f* will copy directly from the original variable to its output signal. For this reason, changing the source f-signal of *push\_f* before *pop\_f* is called is not recommended, and if the instrument instance owning the variable that was passed by *push\_f* is deactivated before *pop\_f* is called, undefined behavior may occur.

Any stack errors (trying to push when there is no more space, or pop from an empty stack, inconsistent number or type of arguments, etc.) are fatal and terminate performance.

## See also

*pop*, *push*, *pop\_f* and *push\_f*.

## Credits

By: Istvan Varga.

2006

# statevar

statevar — State-variable filter.

## Description

Statevar is a new digital implementation of the analogue state-variable filter. This filter has four simultaneous outputs: high-pass, low-pass, band-pass and band-reject. This filter uses oversampling for sharper resonance (default: 3 times oversampling). It includes a resonance limiter that prevents the filter from getting unstable.

## Syntax

```
ahp,alp,abp,abr statevar ain, kcf, kq [, iosamps, istor]
```

## Initialization

*iosamps* -- number of times of oversampling used in the filtering process. This will determine the maximum sharpness of the filter resonance (Q). More oversampling allows higher Qs, less oversampling will limit the resonance. The default is 3 times (iosamps=0).

*istor* --initial disposition of internal data space. Since filtering incorporates a feedback loop of previous output, the initial status of the storage space used is significant. A zero value will clear the space; a non-zero value will allow previous information to remain. The default value is 0.

## Performance

*ahp* -- high-pass output signal.

*alp* -- low-pass output signal.

*abp* -- band-pass signal.

*abr* -- band-reject signal.

*asig* -- input signal.

*kcf* -- filter cutoff frequency

*kq* -- filter Q. This value is limited internally depending on the frequency and the number of times of oversampling used in the process (3-times oversampling by default).

## Examples

### Exemple 504. Example

```
kenv          linseg 0,0.1,1, p3-0.2,1, 0.1, 0
asig          buzz 16000*kenv, 100, 100, 1;
kf            expseg 100, p3/2, 5000, p3/2, 1000
ahp,alp,abp,abr statevar asig, kf, 200
```

outs alp,ahp

## Credits

Author: Victor Lazzarini  
January 2005

New plugin in version 5

January 2005.

# stix

stix — Modèle semi-physique d'un son de baguette.

## Description

*stix* est un modèle semi-physique d'un son de baguette. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares stix iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 30.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,998 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,998 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode stix. Il utilise le fichier *stix.csd* [examples/stix.csd].

### Exemple 505. Exemple de l'opcode stix.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;;RT audio I/O
```

```

; For Non-realtime ouput leave only the line below:
; -o stix.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

;orchestra -----

    sr =          44100
    kr =          4410
    ksmpr =        10
    nchnls =        1

instr 01
a1      line 20, p3, 20      ;an example of stix
a2      stix p4, 0.01      ;preset amplitude increase
a3      product a1, a2      ;stix needs a little amp help at these settings
        out a3              ;increase amplitude
        endin

</CsInstruments>
<CsScore>

;score -----

i1 0 1 26000
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*cabasa, crunch, sandpaper, sekere*

## Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# strchar

strchar — Return the ASCII code of a character in a string

## Description

Return the ASCII code of the character in Sstr at ipos (defaults to zero which means the first character), or zero if ipos is out of range. strchar runs at init time only.

## Syntax

```
ichr strchar Sstr[, ipos]
```

## See also

*strchark*

## Credits

Author: Istvan Varga  
2006

# strchark

strchark — Return the ASCII code of a character in a string

## Description

Return the ASCII code of the character in Sstr at kpos (defaults to zero which means the first character), or zero if kpos is out of range. strchark runs both at init and performance time.

## Syntax

```
kchr strchark Sstr[, kpos]
```

## See also

*strchar*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strcpy

strcpy — Assign value to a string variable

## Description

Assign to a string variable by copying the source which may be a constant or another string variable. strcpy and = copy the string at i-time only.

## Syntax

```
Sdst strcpy Ssrc
```

```
Sdst = Ssrc
```

## Example

```
Sfoo    strcpy "Hello, world !"  
puts Sfoo, 1
```

## See also

*strcpyk*

## Credits

Author: Istvan Varga  
2005



# strcpyk

strcpyk — Assign value to a string variable (k-rate)

## Description

Assign to a string variable by copying the source which may be a constant or another string variable. *strcpyk* does the assignment both at initialization and performance time.

## Syntax

Sdst **strcpyk** Ssrc

## See also

*strcpy*

## Credits

Author: Istvan Varga  
2005

# strcat

strcat — Concatenate strings

## Description

Concatenate two strings and store the result in a variable. *strcat* runs at i-time only. It is allowed for any of the input arguments to be the same as the output variable.

## Syntax

```
Sdst strcat Ssrc1, Ssrc2
```

## Example

```
Sname    = "beats"  
Sname    strcat Sname, ".wav"  
asig     soundin Sname
```

## See also

*strcatk*

## Credits

Author: Istvan Varga  
2005

New in version 5.02

# strcatk

strcatk — Concatenate strings (k-rate)

## Description

Concatenate two strings and store the result in a variable. *strcatk* does the concatenation both at initialization and performance time. It is allowed for any of the input arguments to be the same as the output variable.

## Syntax

```
Sdst strcatk Ssrc1, Ssrc2
```

## See also

*strcat*

## Credits

Author: Istvan Varga  
2005

New in version 5.02

# strcmp

strcmp — Compare strings

## Description

Compare strings and set the result to -1, 0, or 1 if the first string is less than, equal to, or greater than the second, respectively. strcmp compares at i-time only.

## Syntax

```
ires strcmp S1, S2
```

## See also

*strcmpk*

## Credits

Author: Istvan Varga  
2005

# strcmpk

strcmp — Compare strings

## Description

Compare strings and set the result to -1, 0, or 1 if the first string is less than, equal to, or greater than the second, respectively. *strcmpk* does the comparison both at initialization and performance time.

## Syntax

```
kres strcmpk S1, S2
```

## See also

*strcmp*

## Credits

Author: Istvan Varga  
2005

# streson

streson — Résonance d'une corde de fréquence fondamentale variable.

## Description

Un signal audio est modifié par un résonateur de type corde avec une fréquence fondamentale variable.

## Syntaxe

```
ares streson asig, kfrq, ifdbgain
```

## Initialisation

*ifdbgain* -- gain de rétroaction, entre 0 et 1, de la ligne à retard interne. Une valeur proche de 1 crée une décroissance plus lente et une résonance plus prononcée. Avec de petites valeurs, le signal d'entrée peut ne pas être affecté. Dépend de la fréquence du filtre, les valeurs typiques étant > 0.9.

## Exécution

*asig* -- le signal d'entrée audio.

*kfrq* -- la fréquence fondamentale de la corde.

*streson* fait passer l'entrée *asig* à travers un réseau composé de filtres en peigne, passe-bas et passe-tout, comme celui qui est utilisé dans certaines versions de l'algorithme de Karplus-Strong, créant un effet de résonance d'une corde. La fréquence fondamentale de la « corde » est contrôlée par la variable de taux-*k* *kfr*. On peut utiliser cet opcode pour simuler des résonances sympathiques sur un signal d'entrée.

Voir *Rapports de Fréquence Modale* pour les rapports de fréquence d'instruments réels pouvant être utilisés pour déterminer les valeurs de *kfrq*.

*streson* est une adaptation de l'objet *StringFlt* de la bibliothèque d'objets sonores *SndObj* développée par l'auteur.

## Exemples

Voici en exemple de l'opcode *streson*. Il utilise le fichier *streson.csd* [exemples/streson.csd].

### Exemple 506. Exemple de l'opcode streson.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o streson.wav -W ;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a normal sine wave.
asig oscils 8000, 440, 1

; Vary the fundamental frequency of the string
; resonator linearly from 220 to 880 Hertz.
kfr line 220, p3, 880
ifdbgain = 0.95

; Run our sine wave through the string resonator.
astres streson asig, kfr, ifdbgain

; The resonance can get quite loud.
; So we'll clip the signal at 30,000.
al clip astres, 1, 30000
out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for five seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Music Department  
National University of Ireland, Maynooth  
Maynooth, Co. Kildare  
1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.494 de Csound

# strget

strget — Set string variable to value from strset table or string p-field

## Description

*strget* sets a string variable at initialization time to the value stored in *strset* table at the specified index, or a string p-field from the score. If there is no string defined for the index, the variable is set to an empty string.

## Syntax

*Sdst* **strget** *indx*

## Initialization

*indx* -- strset index, or score p-field

*Sdst* -- destination string variable

## See also

*strset*

## Credits

Author: Istvan Varga

2005



# strindex

strindex — Return the position of the first occurrence of a string in another string

## Description

Return the position of the first occurrence of S2 in S1, or -1 if not found. If S2 is empty, 0 is returned. strindex runs at init time only.

## Syntax

```
ipos strindex S1, S2
```

## See also

*strindexk*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strindexk

strindexk — Return the position of the first occurrence of a string in another string

## Description

Return the position of the first occurrence of S2 in S1, or -1 if not found. If S2 is empty, 0 is returned. strindexk runs both at init and performance time.

## Syntax

kpos **strindexk** S1, S2

## See also

*strindex*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strlen

strlen — Return the length of a string

## Description

Return the length of a string, or zero if it is empty. strlen runs at init time only.

## Syntax

```
ilen strlen Sstr
```

## See also

*strlenk*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strlen

strlen — Return the length of a string

## Description

Return the length of a string, or zero if it is empty. strlen runs both at init and performance time.

## Syntax

```
klen strlen Sstr
```

## See also

*strlen*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strlower

strlower — Convert a string to lower case

## Description

Convert Ssrc to lower case, and write the result to Sdst. strlower runs at init time only.

## Syntax

Sdst **strlower** Ssrc

## See also

*strlowerk*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strlowerk

strlowerk — Convert a string to lower case

## Description

Convert Ssrc to lower case, and write the result to Sdst. strlowerk runs both at init and performance time.

## Syntax

```
Sdst strlowerk Ssrc
```

## See also

*strlower*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strrindex

strrindex — Return the position of the last occurrence of a string in another string

## Description

Return the position of the last occurrence of S2 in S1, or -1 if not found. If S2 is empty, the length of S1 is returned. strrindex runs at init time only.

## Syntax

`ipos strrindex S1, S2`

## See also

*strrindexk*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strrindexk

strrindexk — Return the position of the last occurrence of a string in another string

## Description

Return the position of the last occurrence of S2 in S1, or -1 if not found. If S2 is empty, the length of S1 is returned. strrindexk runs both at init and performance time.

## Syntax

```
kpos strrindexk S1, S2
```

## See also

*strrindex*

## Credits

Author: Istvan Varga  
2006

New in version 5.02



# strset

strset — Allows a string to be linked with a numeric value.

## Description

Allows a string to be linked with a numeric value.

## Syntax

```
strset iarg, istring
```

## Initialization

*iarg* -- the numeric value.

*istring* -- the alphanumeric string (in double-quotes).

*strset* (optional) allows a string, such as a filename, to be linked with a numeric value. Its use is optional.

## Examples

The following statement, used in the orchestra header, will allow the numeric value 10 to be substituted anywhere the soundfile *asound.wav* is called for.

```
strset 10, "asound.wav"
```

## Examples

Here is an example of the strset opcode. It uses the file *strset.csd* [examples/strset.csd].

### Exemple 507. Example of the strset opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac       -iadc      -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 1
nchnls = 1

;Example by Andres Cabrera 2008
```

```
; \\n is used to denote "new line"
strset 1, "String 1\\n"
strset 2, "String 2\\n"

instr 1
Str strget p4
prints Str
endin

</CsInstruments>
<CsScore>
;      p4 is used to select string
i 1 0 1 1
i 1 3 1 2
</CsScore>
</CsoundSynthesizer>
```

## See Also

*pset* and *strget*

# strsub

strsub — Extract a substring

## Description

Return a substring of the source string. strsub runs at init time only.

## Syntax

```
Sdst strsub Ssrc[, istart[, iend]]
```

## Initialization

*istart* (optional, defaults to 0) -- start position in Ssrc, counting from 0. A negative value means the end of the string.

*iend* (optional, defaults to -1) -- end position in Ssrc, counting from 0. A negative value means the end of the string. If iend is less than istart, the output is reversed.

## Examples

Here is an example of the strsub opcode. It uses the file *strsub.csd* [examples/strsub.csd].

### Exemple 508. Example of the strsub opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac -iadc ;;;-d RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o strsub.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; By: Jonathan Murphy 2007

instr 1
  Smember strget p4

  ; Parse Smember
  istrlen strlen Smember
  idelimiter strindex Smember, ":"

  S1 strsub Smember, 0, idelimiter ; "String1"
  S2 strsub Smember, idelimiter + 1, istrlen ; "String2"

  printf "First string: %s\nSecond string: %s\n", 1, S1, S2

endin

</CsInstruments>
<CsScore>
i 1 0 1 "String1:String2"
</CsScore>
</CsoundSynthesizer>
```

## See also

*strsubk*

## Credits

Author: Istvan Varga  
2006

# strsubk

strsubk — Extract a substring

## Description

Return a substring of the source string. strsubk runs both at init and performance time.

## Syntax

Sdst **strsubk** Ssrc, kstart, kend

## Performance

*kstart* -- start position in Ssrc, counting from 0. A negative value means the end of the string.

*kend* -- end position in Ssrc, counting from 0. A negative value means the end of the string. If *kend* is less than *kstart*, the output is reversed.

## See also

*strsub*

## Credits

Author: Istvan Varga  
2006

# strtod

strtod — Converts a string to a float (i-rate).

## Description

Convert a string to a floating point value. It is also possible to pass an strset index or a string p-field from the score instead of a string argument. If the string cannot be parsed as a floating point or integer number, an init or perf error occurs and the instrument is deactivated.

## Syntax

```
ir strtod Sstr
```

```
ir strtod indx
```

## Initialization

*Sstr* -- String to convert.

*indx* -- index of string set by strset

## Performance

*ir* -- Value of string as float.

## Credits

Author: Istvan Varga  
2005

# strtodk

strtodk — Converts a string to a float (k-rate).

## Description

Convert a string to a floating point value at i- or k-rate. It is also possible to pass an strset index or a string p-field from the score instead of a string argument. If the string cannot be parsed as a floating point or integer number, an init or perf error occurs and the instrument is deactivated.



### Note

If a k-rate index variable is used, it should be valid at i-time as well.

## Syntax

```
kr strtodk Sstr
```

```
kr strtodk kndx
```

## Performance

*kr* -- Value of string as float.

*Sstr* -- String to convert.

*indx* -- index of string set by strset

## Credits

Author: Istvan Varga  
2005

# strtol

strtol — Converts a string to a signed integer (i-rate).

## Description

Convert a string to a signed integer value. It is also possible to pass an strset index or a string p-field from the score instead of a string argument. If the string cannot be parsed as a floating point or integer number, an init or perf error occurs and the instrument is deactivated.

## Syntax

```
ir strtol Sstr
```

```
ir strtol indx
```

## Initialization

*Sstr* -- String to convert.

*indx* -- index of string set by strset

strtol can parse numbers in decimal, octal (prefixed by 0), and hexadecimal (with a prefix of 0x) format.

## Performance

*ir* -- Value of string as signed integer.

## Credits

Author: Istvan Varga  
2005



# strtolk

strtolk — Converts a string to a signed integer (k-rate).

## Description

Convert a string to a floating point value at i- or k-rate. It is also possible to pass an strset index or a string p-field from the score instead of a string argument. If the string cannot be parsed as a floating point or integer number, an init or perf error occurs and the instrument is deactivated.



### Note

If a k-rate index variable is used, it should be valid at i-time as well.

## Syntax

```
kr strtolk Sstr
```

```
kr strtolk kndx
```

strtolk can parse numbers in decimal, octal (prefixed by 0), and hexadecimal (with a prefix of 0x) format.

## Performance

*kr* -- Value of string as signed integer.

*Sstr* -- String to convert.

*indx* -- index of string set by strset

## Credits

Author: Istvan Varga  
2005

# strupper

strupper — Convert a string to upper case

## Description

Convert Ssrc to upper case, and write the result to Sdst. strupper runs at init time only.

## Syntax

```
Sdst strupper Ssrc
```

## See also

*strupperk*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# strupperk

strupperk — Convert a string to upper case

## Description

Convert Ssrc to upper case, and write the result to Sdst. strupperk runs both at init and performance time.

## Syntax

Sdst **strupperk** Ssrc

## See also

*strupper*

## Credits

Author: Istvan Varga  
2006

New in version 5.02

# subinstr

subinstr — Creates and runs a numbered instrument instance.

## Description

Creates an instance of another instrument and is used as if it were an opcode.

## Syntax

```
a1, [...] [, a8] subinstr instrnum [, p4] [, p5] [...]
```

```
a1, [...] [, a8] subinstr "insname" [, p4] [, p5] [...]
```

## Initialization

*instrnum* -- Number of the instrument to be called.

« *insname* » -- A string (in double-quotes) representing a named instrument.

For more information about specifying input and output interfaces, see *Calling an Instrument within an Instrument*.

## Performance

*a1*, ..., *a8* -- The audio output from the called instrument. This is generated using the *signal output* opcodes.

*p4*, *p5*, ... -- Additional input values the are mapped to the called instrument p-fields, starting with *p4*.

The called instrument's *p2* and *p3* values will be identical to the host instrument's values. While the host instrument can *control its own duration*, any such attempts inside the called instrument will most likely have no effect.

## See Also

*Calling an Instrument within an Instrument*, *event*, *schedule*, *subinstrinit*

## Examples

Here is an example of the subinstr opcode. It uses the file *subinstr.csd* [examples/subinstr.csd].

### Exemple 509. Example of the subinstr opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o subinstr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - Creates a basic tone.
instr 1
; Print the value of p4, should be equal to
; Instrument #2's iamp field.
print p4

; Print the value of p5, should be equal to
; Instrument #2's ipitch field.
print p5

; Create a tone.
asig oscils p4, p5, 0

out asig
endin

; Instrument #2 - Demonstrates the subinstr opcode.
instr 2
iamp = 20000
ipitch = 440

; Use Instrument #1 to create a basic sine-wave tone.
; Its p4 parameter will be set using the iamp variable.
; Its p5 parameter will be set using the ipitch variable.
abasic subinstr 1, iamp, ipitch

; Output the basic tone that we have created.
out abasic
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Here is an example of the subinstr opcode using a named instrument. It uses the file *subinstr\_named.csd* [examples/subinstr\_named.csd].

### Exemple 510. Example of the subinstr opcode using a named instrument.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o subinstr_named.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument "basic_tone" - Creates a basic tone.
instr basic_tone
; Print the value of p4, should be equal to
; Instrument #2's iamp field.
print p4

; Print the value of p5, should be equal to
; Instrument #2's ipitch field.
print p5

; Create a tone.
asig oscils p4, p5, 0

out asig
endin

; Instrument #1 - Demonstrates the subinstr opcode.
instr 1
iamp = 20000
ipitch = 440

; Use the "basic_tone" named instrument to create a
; basic sine-wave tone.
; Its p4 parameter will be set using the iamp variable.
; Its p5 parameter will be set using the ipitch variable.
abasic subinstr "basic_tone", iamp, ipitch

; Output the basic tone that we have created.
out abasic
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

New in version 4.21

# subinstrinit

subinstrinit — Creates and runs a numbered instrument instance at init-time.

## Description

Same as *subinstr*, but init-time only and has no output arguments.

## Syntax

```
subinstrinit instrnum [, p4] [, p5] [...]
```

```
subinstrinit "insname" [, p4] [, p5] [...]
```

## Initialization

*instrnum* -- Number of the instrument to be called.

« *insname* » -- A string (in double-quotes) representing a named instrument.

For more information about specifying input and output interfaces, see *Calling an Instrument within an Instrument*.

## Performance

*p4*, *p5*, ... -- Additional input values the are mapped to the called instrument p-fields, starting with *p4*.

The called instrument's *p2* and *p3* values will be identical to the host instrument's values. While the host instrument can *control its own duration*, any such attempts inside the called instrument will most likely have no effect.

## See Also

*Calling an Instrument within an Instrument*, *event*, *schedule*, *subinstr*

## Credits

New in version 4.23

## sum

sum — Somme de n'importe quel nombre de signaux de taux-a.

## Description

Somme de n'importe quel nombre de signaux de taux-a.

## Syntaxe

```
ares sum asig1 [, asig2] [, asig3] [...]
```

## Exécution

*asig1, asig2, ...* -- signaux de taux-a à additionner (à mélanger).

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Avril 1999

Nouveau dans le version 3.54 de Csound



# svfilter

svfilter — A resonant second order filter, with simultaneous lowpass, highpass and bandpass outputs.

## Description

Implementation of a resonant second order filter, with simultaneous lowpass, highpass and bandpass outputs.

## Syntax

```
alow, ahigh, aband svfilter asig, kcf, kq [, iscl]
```

## Initialization

*iscl* -- coded scaling factor, similar to that in *reson*. A non-zero value signifies a peak response factor of 1, i.e. all frequencies other than *kcf* are attenuated in accordance with the (normalized) response curve. A zero value signifies no scaling of the signal, leaving that to some later adjustment (see *balance*). The default value is 0.

## Performance

*svfilter* is a second order state-variable filter, with k-rate controls for cutoff frequency and Q. As Q is increased, a resonant peak forms around the cutoff frequency. *svfilter* has simultaneous lowpass, highpass, and bandpass filter outputs; by mixing the outputs together, a variety of frequency responses can be generated. The state-variable filter, or "multimode" filter was a common feature in early analog synthesizers, due to the wide variety of sounds available from the interaction between cutoff, resonance, and output mix ratios. *svfilter* is well suited to the emulation of "analog" sounds, as well as other applications where resonant filters are called for.

*asig* -- Input signal to be filtered.

*kcf* -- Cutoff or resonant frequency of the filter, measured in Hz.

*kq* -- Q of the filter, which is defined (for bandpass filters) as bandwidth/cutoff. *kq* should be in a range between 1 and 500. As *kq* is increased, the resonance of the filter increases, which corresponds to an increase in the magnitude and "sharpness" of the resonant peak. When using *svfilter* without any scaling of the signal (where *iscl* is either absent or 0), the volume of the resonant peak increases as Q increases. For high values of Q, it is recommended that *iscl* be set to a non-zero value, or that an external scaling function such as *balance* is used.

*svfilter* is based upon an algorithm in Hal Chamberlin's *Musical Applications of Microprocessors* (Hayden Books, 1985).

## Examples

Here is an example of the svfilter opcode. It uses the file *svfilter.csd* [examples/svfilter.csd].

### Exemple 511. Example of the svfilter opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o svfilter.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
; The separate outputs of the filter are scaled by values from the score,
; and are mixed together.
sr = 44100
kr = 2205
ksmps = 20
nchnls = 1

instr 1

    idur      = p3
    ifreq     = p4
    iamp      = p5
    ilowamp   = p6           ; determines amount of lowpass output in signal
    ihighamp  = p7           ; determines amount of highpass output in signal
    ibandamp  = p8           ; determines amount of bandpass output in signal
    iq        = p9           ; value of q

    iharms    =      (sr*.4) / ifreq

    asig      gbuzz 1, ifreq, iharms, 1, .9, 1           ; Sawtooth-like waveform
    kfreq     linseg 1, idur * 0.5, 4000, idur * 0.5, 1 ; Envelope to control filter cutoff

    alow, ahigh, aband    svfilter asig, kfreq, iq

    aout1     =      alow * ilowamp
    aout2     =      ahigh * ihighamp
    aout3     =      aband * ibandamp
    asum      =      aout1 + aout2 + aout3
    kenv      linseg 0, .1, iamp, idur -.2, iamp, .1, 0 ; Simple amplitude envelope
    out       asum * kenv

endin

</CsInstruments>
<CsScore>

f1 0 8192 9 1 1 .25

i1 0 5 100 1000 1 0 0 5 ; lowpass sweep
i1 5 5 200 1000 1 0 0 30 ; lowpass sweep, octave higher, higher q
i1 10 5 100 1000 0 1 0 5 ; highpass sweep
i1 15 5 200 1000 0 1 0 30 ; highpass sweep, octave higher, higher q
i1 20 5 100 1000 0 0 1 5 ; bandpass sweep
i1 25 5 200 1000 0 0 1 30 ; bandpass sweep, octave higher, higher q
i1 30 5 200 2000 .4 .6 0 ; notch sweep - notch formed by combining highpass and lowpass outputs
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Sean Costello  
 Seattle, Washington  
 1999

New in Csound version 3.55

# syncgrain

syncgrain — Synthèse granulaire synchrone.

## Description

*syncgrain* implémente la synthèse granulaire synchrone. La source de son pour les grains est obtenue par la lecture d'une table de fonction contenant les échantillons de la forme d'onde source. Pour les sources de son échantillonné, on utilise *GEN01*. *syncgrain* acceptera des tables allouées en différé.

Le générateur de grain exerce un contrôle total sur la fréquence (grains/sec), l'amplitude générale, la hauteur du grain (incrément d'échantillonnage) et la taille du grain (en sec), comme paramètres constants ou variant dans le temps (signaux). Le taux du pointeur de grain est un paramètre supplémentaire qui contrôle à quelle position le générateur commencera à lire les échantillons dans la table pour chaque grain successif. Il est mesuré en fraction de la taille du grain ; s'il vaut 1 (la valeur par défaut) chaque grain successif est lu à partir de l'endroit où le grain précédent s'est terminé. S'il vaut 0,5 le grain suivant commencera à mi-chemin entre la position de début et la position de fin du grain précédent, etc. S'il vaut 0 le générateur lira toujours à partir de la même position dans la table (quelque soit l'endroit où le pointeur se trouvait juste avant). Avec une valeur négative le pointeur évoluera en décrémentant sa position. Ce contrôle apporte plus de flexibilité dans la création de modifications de l'échelle temporelle lors de la resynthèse.

*syncgrain* générera n'importe quel nombre de flux parallèles de grains (en fonction de la densité/fréquence de grains), borné supérieurement par la valeur de *iolaps* (100 par défaut). Le nombre de flux (grains se chevauchant) est déterminé par *taille\_du\_grain\*fréquence\_du\_grain*. Plus il y aura de chevauchements de grains, plus il y aura de calculs et il se peut que la synthèse ne s'effectue pas en temps réel (cela dépend de la puissance du processeur).

*syncgrain* peut simuler une synthèse formantique à la FOF, si l'on utilise une forme d'enveloppe de grain adéquate et une sinusoïde comme forme d'onde du grain. Dans ce cas, on pourra utiliser des tailles de grain d'environ 0,04 sec. La fréquence centrale du formant est déterminée par la hauteur du grain. Comme l'incrément est en échantillons, si l'on veut utiliser une fréquence en Hz, cette valeur doit être multipliée par *taille\_de\_la\_table/sr*. La fréquence du grain détermine le fondamental.

*syncgrain* utilise des indices en virgule flottante, ce qui fait qu'il n'est pas affecté par des tables de grande taille. Cet opcode est basé sur la class *SyncGrain* de la bibliothèque *SndObj*.

## Syntaxe

```
asig syncgrain kamp, kfreq, kpitch, kgrsize, kprate, ifun1, \  
      ifun2, iolaps
```

## Initialisation

*ifun1* -- table de fonction du signal source. Des tables avec allocation différée sont acceptées (voir *GEN01*), mais l'opcode attend une source mono.

*ifun2* -- table de fonction de l'enveloppe du grain.

*iolaps* -- nombre maximum de chevauchements,  $\max(kfreq) \times \max(kgrsize)$ . Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

## Exécution

*kamp* -- pondération de l'amplitude.

*kfreq* -- fréquence de génération des grains, ou densité, en grains/sec.

*kpitch* -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers).

*kgrsize* -- taille du grain en secondes.

*kprate* -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

## Exemples

### Exemple 512. Exemple

```
iolaps = 2
igrsize = 0.04
ifreq = iolaps/igrsize
ips = 1/iolaps

istr = .5 /* timescale */
ipitch = 1 /* pitchscale */

a1 syncgrain 16000, ifreq, ipitch, igrsize, ips*istr, 1, 2, iolaps
out a1
```

## Crédits

Auteur: Victor Lazzarini  
Janvier 2005

Nouveau plugin dans la version 5

Janvier 2005.

# syncloop

syncloop — Synthèse granulaire synchrone.

## Description

*syncloop* est une variation sur *syncgrain*, qui implémente la synthèse granulaire synchrone. *syncloop* ajoute des points de début et de fin de boucle et une position de départ facultative. Le début et la fin de boucle contrôlent les positions de démarrage des grains, si bien que les grains réalisés peuvent s'étendre au-delà des points de la boucle (si les points de la boucle ne sont pas aux extrémités de la table), ce qui permet des transitions fluides. Pour plus d'information sur le procédé de synthèse granulaire, voir la page du manuel sur *syncgrain*.

## Syntaxe

```
asig syncloop kamp, kfreq, kpitch, kgrsize, kprate, klstart, \  
      klend, ifun1, ifun2, iolaps[,istart, iskip]
```

## Initialisation

*ifun1* -- table de fonction du signal source. Des tables avec allocation différée sont acceptées (voir *GEN01*), mais l'opcode attend une source mono.

*ifun2* -- table de fonction de l'enveloppe du grain.

*iolaps* -- nombre maximum de chevauchements,  $\max(kfreq) \cdot \max(kgrsize)$ . Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

*istart* -- point de départ de la synthèse en secs (0 par défaut).

*iskip* -- s'il vaut 1, l'initialisation de l'opcode est ignorée, pour les notes liées, l'exécution continuant depuis la position à l'intérieur de la boucle où la note précédente s'est terminée. La valeur par défaut de 0 signifie que l'initialisation n'est pas ignorée.

## Exécution

*kamp* -- pondération de l'amplitude.

*kfreq* -- fréquence de génération des grains, ou densité, en grains/sec.

*kpitch* -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers).

*kgrsize* -- taille du grain en secondes.

*kprate* -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

*klstart* -- début de la boucle en secs.

*klend* -- fin de la boucle en secs.

## Exemples

### Exemple 513. Exemple

```
iolaps = 2
igrsize = 0.04
ifreq = iolaps/igrsize
ips = 1/iolaps

istr = .5 /* timescale */
ipitch = 1 /* pitchscale */

a1 syncloop 16000, ifreq, ipitch, igrsize, ips*istr, 1, 2, 1, 2, iolaps
out a1
```

## Crédits

Auteur : Victor Lazzarini  
Janvier 2005

Nouveau plugin dans la version 5

Janvier 2005.

# syncphasor

syncphasor — Produit une valeur de phase mobile normalisée avec entrée et sortie de synchronisation.

## Description

Produit une valeur de phase mobile entre zéro et un et une impulsion supplémentaire en sortie ("sync out") chaque fois que sa valeur de phase traverse le zéro ou est remise à zéro. La phase peut être réinitialisée à tout instant par une impulsion sur le paramètre "sync in".

## Syntaxe

```
aphase, asyncout syncphasor xcps, asyncin, [, iphs]
```

## Initialisation

*iphs* (facultatif) -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

## Exécution

*aphase* -- la valeur de phase en sortie ; toujours entre 0 et 1.

*asyncout* -- la sortie de synchronisation prend la valeur 1.0 durant un échantillon chaque fois que la valeur de phase traverse le zéro ou que l'entrée de synchronisation a une valeur non nulle. Elle vaut zéro aux autres moments.

*asyncin* -- l'entrée de synchronisation provoque la remise à zéro de la phase chaque fois que *asyncin* est non nul.

*xcps* -- fréquence du phaseur en Hertz. Si *xcps* est négatif, la phase sera décrémentée de 1 à 0 au lieu d'être incrémentée.

Une phase interne est augmentée successivement selon la fréquence de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle  $0 \leq \text{phs} < 1$ . Lorsqu'elle est utilisée comme indice dans une *table*, cette phase (multipliée par la longueur de la table de fonction) permettra de l'utiliser comme un oscillateur.

La phase de *syncphasor* peut être synchronisée à un autre phaseur (ou à un autre signal) au moyen du paramètre *asyncin*. Chaque fois que *asyncin* prend une valeur non nulle, la valeur de *aphase* est remise à zéro. *syncphasor* sort aussi son propre signal de "synchro" qui consiste en une impulsion d'un échantillon chaque fois que sa phase traverse le zéro ou est réinitialisée. On peut ainsi facilement mettre en série plusieurs opcodes *syncphasor* pour créer un effet d'oscillateur "hard sync".

## Exemples

Voici un exemple de l'opcode syncphasor. Il utilise le fichier *syncphasor.csd* [examples/syncphasor.csd].

### Exemple 514. Exemple de l'opcode syncphasor.



Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

instr 1
; Use two syncphasors - one is the "master",
; the other the "slave"

; master's frequency determines pitch
imastercps =      cpspch(p4)
imaxamp    =      10000

; the slave's frequency affects the timbre
kslavecps  line    imastercps, p3, imastercps * 3

; the master "oscillator"
; the master has no sync input
anosync    init    0.0
am, async  syncphasor imastercps, anosync

; the slave "oscillator"
aout, as   syncphasor kslavecps, async

adeclick   linseg   0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

; Output the slave's phase value which is a rising
; sawtooth wave. This produces aliasing, but hey, this
; this is just an example ;)

      out          aout * adeclick * imaxamp

endin

</CsInstruments>
<CsScore>

i1 0 1      7.00
i1 + 0.5    7.02
i1 + .      7.05
i1 + .      7.07
i1 + .      7.09
i1 + 2      7.06

e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode syncphasor. Il utilise le fichier *syncphasor-CZresonance.csd* [examples/syncphasor-CZresonance.csd].

### Exemple 515. Un autre exemple de l'opcode syncphasor.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o syncphasor-CZresonance.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; by Anthony Kozar. February 2008
```

```
; http://www.anthonykozar.net/

; Imitation of the Casio CZ-series synthesizer's "Resonance" waveforms
; using a synced phasor to read a sinusoid table. The jumps at the sync
; points are smoothed by multiplying with a windowing function controlled
; by the master phasor.

; Based on information from the Wikipedia article on phase distortion:
; http://en.wikipedia.org/wiki/Phase_distortion_synthesis

; Sawtooth Resonance waveform. Smoothing function is just the inverted
; master phasor.

; The Wikipedia article shows an inverted cosine as the stored waveform,
; which implies that it must be unipolar for the smoothing to work.
; I have substituted a sine wave in the first phrase to keep the output
; bipolar. The second phrase demonstrates the much "rezzier" sound of the
; bipolar cosine due to discontinuities.

instr 1
  ifreq      =      cpspch(p4)
  initReson  =      p5
  itable     =      p6
  imaxamp    =      10000
  anosync    init  0.0

  kslavecps  line    ifreq * initReson, p3, ifreq
  amaster, async syncphasor ifreq, anosync      ; pair of phasors
  aslave, async2 syncphasor kslavecps, async    ; slave synced to master
  aosc       tablei  aslave, itable, 1          ; use slave phasor to read a (co)sine table
  aout       =       aosc * (1.0 - amaster) ; inverted master smoothes jumps
  adeclick   linseg  0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

                      out      aout * adeclick * imaxamp

endin

; Triangle or Trapezoidal Resonance waveform. Uses a second table to change
; the shape of the smoothing function. (This is my best guess so far as to
; how these worked). The cosine table works fine with the triangular smoothing
; but we once again need to use a sine table with the trapezoidal smoothing.

; (It might be interesting to be able to vary the "width" of the trapezoid.
; This could be done with the pdhalf opcode).

instr 2
  ifreq      =      cpspch(p4)
  initReson  =      p5
  itable     =      p6
  ismoothtbl =      p7
  imaxamp    =      10000
  anosync    init  0.0

  kslavecps  line    ifreq * initReson, p3, ifreq
  amaster, async syncphasor ifreq, anosync      ; pair of phasors
  aslave, async2 syncphasor kslavecps, async    ; slave synced to master
  aosc       tablei  aslave, itable, 1          ; use slave phasor to read a (co)sine table
  asmooth    tablei  amaster, ismoothtbl, 1 ; use master phasor to read smoothing table
  aout       =       aosc * asmooth
  adeclick   linseg  0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

                      out      aout * adeclick * imaxamp

endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1
f3 0 16385 9 1 1 270 ; inverted cosine
f5 0 4097 7 0.0 2048 1.0 2049 0.0 ; unipolar triangle
f6 0 4097 7 1.0 2048 1.0 2049 0.0 ; "trapezoid"

; Sawtooth resonance with a sine table
i1 0 1 7.00 5.0 1
i. + 0.5 7.02 4.0
i. + . 7.05 3.0
i. + . 7.07 2.0
i. + . 7.09 1.0
i. + 2 7.06 12.0
f0 6
s

; Sawtooth resonance with a cosine table
```

```

i1 0 1      7.00  5.0  3
i. + 0.5    7.02  4.0
i. + .      7.05  3.0
i. + .      7.07  2.0
i. + .      7.09  1.0
i. + 2      7.06 12.0
f0 6
s

; Triangle resonance with a cosine table
i2 0 1      7.00  5.0  3  5
i. + 0.5    7.02  4.0
i. + .      7.05  3.0
i. + .      7.07  2.0
i. + .      7.09  1.0
i. + 2      7.06 12.0
f0 6
s

; Trapezoidal resonance with a sine table
i2 0 1      7.00  5.0  1  6
i. + 0.5    7.02  4.0
i. + .      7.05  3.0
i. + .      7.07  2.0
i. + .      7.09  1.0
i. + 2      7.06 12.0

e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*phasor*.

Et les opcodes d'Accès aux Table comme : *table*, *tablei*, *table3* et *tab*.

## Crédits

Adapté d'après l'opcode *phasor* par Anthony Kozar  
Janvier 2008

Nouveau dans la version 5.08 de Csound

# system

system — Call an external program via the system call

## Description

**system** and **system\_i** call any external command understood by the operating system, similarly to the C function `system()`. **system\_i** runs at i-time only, while **system** runs both at initialization and performance time.

## Syntax

```
ires system_i itrig, Scmd, [inowait]
```

```
kres system ktrig, Scmd, [knowait]
```

## Initialization

*Scmd* -- command string

*itrig* -- if greater than zero the opcode performs the printing; otherwise it is a null operation.

## Performance

*ktrig* -- if greater than zero and different from the value on the previous control cycle the opcode performs the requested printing. Initially this previous value is taken as zero.

*inowait, knowait* -- if given a non zero the command is run in the background and the command does not wait for the result. (default = 0)

*ires, kres* -- the return code of the command in wait mode and if the command is run. In other cases returns zero.

More than one system command (a script) can be executed with a single **system** opcode by using double braces strings `{ { } }`.



### Note

This opcode is very system dependant, so should be used with extreme care (or not used) if platform neutrality is desired.

## Example

Here is an example of the `system_i` opcode. It uses the file `system.csd` [examples/system.csd].

### Exemple 516. Example of the system opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            ; -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o system.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
; Waits for command to execute before continuing
ires system_i 1,{{      ps
                        date
                        cd ~/Desktop
                        pwd
                        ls -l
                        whois csounds.com
                        }}
print ires
turnoff
endin

instr 2
; Runs command in a separate thread
ires system_i 1,{{      ps
                        date
                        cd ~/Desktop
                        pwd
                        ls -l
                        whois csounds.com
                        }}, 1

print ires
turnoff
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for thirty seconds.
i 1 0 1
i 2 5 1
e

</CsScore>
</CsoundSynthesizer>

```

## Credits

Author: John fitch  
2007

New in version 5.06

## tb

tb0, tb1, tb2, tb3, tb4, tb5, tb6, tb7, tb8, tb9, tb10, tb11, tb12, tb13, tb14, tb15, tb0\_init, tb1\_init, tb2\_init, tb3\_init, tb4\_init, tb5\_init, tb6\_init, tb7\_init, tb8\_init, tb9\_init, tb10\_init, tb11\_init, tb12\_init, tb13\_init, tb14\_init, tb15\_init — Accès en lecture à une table depuis une expression.

## Description

Permet de lire des tables de manière fonctionnelle, à utiliser dans des expressions. Actuellement, Csound ne supporte que les fonctions avec un seul argument en entrée. Cependant, pour accéder aux éléments d'une table, on doit fournir deux nombres : le numéro de la table et l'indice de l'élément. Donc, afin de pouvoir accéder à un élément d'une table par une fonction, il faut une étape de préparation.

## Syntaxe

```
tb0_init ifn
```

```
tb1_init ifn
```

```
tb2_init ifn
```

```
tb3_init ifn
```

```
tb4_init ifn
```

```
tb5_init ifn
```

```
tb6_init ifn
```

```
tb7_init ifn
```

```
tb8_init ifn
```

```
tb9_init ifn
```

```
tb10_init ifn
```

```
tb11_init ifn
```

```
tb12_init ifn
```

```
tb13_init ifn
```

```
tb14_init ifn
```

```
tb15_init ifn
```

```
iout = tb0(iIndex)
```

```
kout = tb0(kIndex)
```

```
iout = tb1(iIndex)

kout = tb1(kIndex)

iout = tb2(iIndex)

kout = tb2(kIndex)

iout = tb3(iIndex)

kout = tb3(kIndex)

iout = tb4(iIndex)

kout = tb4(kIndex)

iout = tb5(iIndex)

kout = tb5(kIndex)

iout = tb6(iIndex)

kout = tb6(kIndex)

iout = tb7(iIndex)

kout = tb7(kIndex)

iout = tb8(iIndex)

kout = tb8(kIndex)

iout = tb9(iIndex)

kout = tb9(kIndex)

iout = tb10(iIndex)

kout = tb10(kIndex)

iout = tb11(iIndex)

kout = tb11(kIndex)

iout = tb12(iIndex)

kout = tb12(kIndex)

iout = tb13(iIndex)

kout = tb13(kIndex)

iout = tb14(iIndex)
```

```
kout = tb14(kIndex)
```

```
iout = tb15(iIndex)
```

```
kout = tb15(kIndex)
```

## Exécution

Il y a 16 opcodes différents dont le nom est associé à un nombre compris entre 0 et 15. Il faut associer une table spécifique avec chaque opcode (ainsi le nombre maximum de tables accessibles de manière fonctionnelle est 16). Avant de pouvoir accéder à une table, celle-ci doit être associée avec l'un des 16 opcodes au moyen d'un opcode choisi parmi *tb0\_init*, ..., *tb15\_init*. Par exemple,

```
tb0_init 1
```

associe la table 1 avec la fonction *tb0*( ), si bien que chaque élément de la table 1 peut être atteint (de manière fonctionnelle) par :

```
kvar = tb0(k_some_index_of_table1) * k_some_other_var
```

```
ivar = tb0(i_some_index_of_table1) + i_some_other_var
```

etc...

En utilisant ces opcodes, on peut réduire considérablement le nombre de lignes d'un orchestre, ce qui améliore sa lisibilité.

## Crédits

Ecrit par Gabriel Maldonado.



# tab

tab — Opcodes de table rapides.

## Description

Opcodes de table rapides. Plus rapides que *table* et que *tablew* parce qu'ils fonctionnent sans indexation cyclique et sans limite et qu'ils ne testent pas la validité des index. Ils ont été implémentés pour fournir un accès rapide aux tableaux. Ils supportent les tables dont la longueur n'est pas une puissance de deux (pouvant être générées par n'importe quelle fonction GEN en lui donnant une valeur de longueur négative).

## Syntaxe

```
ir tab_i indx, ifn[, ixmode]

kr tab kndx, ifn[, ixmode]

ar tab xndx, ifn[, ixmode]

tabw_i isig, indx, ifn [,ixmode]

tabw ksig, kndx, ifn [,ixmode]

tabw asig, andx, ifn [,ixmode]
```

## Initialisation

*ifn* -- numéro de la table.

*ixmode* -- zéro par défaut. S'il est nul, l'intervalle de variation de *xndx* et de *ixoff* est la longueur de la table ; s'il est non nul, *xndx* et *ixoff* varient entre 0 et 1.

*isig* -- valeur d'entrée à écrire.

*indx* -- index de la table.

## Exécution

*asig*, *ksig* -- signal d'entrée à écrire.

*andx*, *kndx* -- index de la table.

Les opcodes *tab* et *tabw* sont semblables à *table* et à *tablew*, mais ils sont plus rapides et supportent des tables dont la longueur n'est une puissance de deux.

Il faut apporter une attention spéciale aux valeurs de l'index. Des valeurs d'index en dehors de l'espace alloué pour la table provoqueront un plantage de Csound.

## Crédits

Ecrit par Gabriel Maldonado.

# tabrec

tabrec — Recording of control signals.

## Description

Records control-rate signals on trigger-temporization basis.

## Syntax

```
tabrec   ktrig_start, ktrig_stop, knumtics, kfn, kin1 [,kin2,...,kinN]
```

## Performance

*ktrig\_start* -- start recording when non-zero.

*ktrig\_stop* -- stop recording when knumtics trigger impulses are received by this input argument.

*knumtics* -- stop recording or reset playing pointer to zero when the number of tics defined by this argument is reached.

*kfn* -- table where k-rate signals are recorded.

*kin1,...,kinN* -- input signals to record.

The *tabrec* and *tabplay* opcodes allow to record/playback control signals on trigger-temporization basis.

*tabrec* opcode records a group of k-rate signals by storing them into kfn table. Each time *ktrig\_start* is triggered, *tabrec* resets the table pointer to zero and begins to record. Recording phase stops after knumtics trigger impluses have been received by *ktrig\_stop* argument.

These opcodes can be used like a sort of ``middle-term" memory that ``remembers" generated signals. Such memory can be used to supply generative music with a coherent iterative compositional structure.

## See Also

*tabplay*

## Credits

Written by Gabriel Maldonado.

# table

table — Accède aux valeurs d'une table par indexation directe.

## Description

Accède aux valeurs d'une table par indexation directe.

## Syntaxe

```
ares table andx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ires table indx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
kres table kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction.

*ixmode* (facultatif) -- type de l'index. La valeur par défaut est 0.

- 0 = index brut
- 1 = normalisé (de 0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est au centre, utiliser *taille\_table/2* (brut) ou 0.5 (normalisé). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = indexation normale (index < 0 traité comme index=0 ; index > *taille\_table* ramené à index=*taille\_table*)
- 1 = indexation cyclique.

## Exécution

*table* effectue une consultation de table avec des index variant au taux d'initialisation, de contrôle ou audio. Ces index peuvent être des nombres bruts (0, 1, 2, ..., *taille* - 1) ou des valeurs normalisées (0 à 1). Les index sont d'abord modifiés par la valeur de décalage puis leur appartenance à un intervalle valable est testée avant la consultation de la table (voir *iwrap*). Si l'index peut prendre la valeur maximale ou si l'on utilise l'interpolation, la table doit avoir un point de garde. Une *table* indexée par un phaseur périodique (voir *phasor*) simulera un oscillateur.

## Exemples

Voici un exemple de l'opcode *table*. Il utilise le fichier *table.csd* [examples/table.csd].

## Exemple 517. Exemple de l'opcode table.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o table.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Vary our index linearly from 0 to 1.
kndx line 0, p3, 1

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kfreq table kndx, ifn, ixmode

; Generate a sine waveform, use our table values
; to vary its frequency.
a1 oscil 20000, kfreq, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a line from 200 to 2,000.
f 1 0 1025 -7 200 1024 2000
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*tablei, table3, oscil1, oscil1i, osciln*

## Crédits

Exemple écrit par Kevin Conder.

# table3

table3 — Accède aux valeurs d'une table par indexation directe avec interpolation cubique.

## Description

Accède aux valeurs d'une table par indexation directe avec interpolation cubique.

## Syntaxe

```
ares table3 andx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ires table3 indx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
kres table3 kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction.

*ixmode* (facultatif) -- type de l'index. La valeur par défaut est 0.

- 0 = index brut
- 1 = normalisé (de 0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est au centre, utiliser *taille\_table/2* (brut) ou 0.5 (normalisé). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = indexation normale (index < 0 traité comme index=0 ; index > *taille\_table* ramené à index=*taille\_table*)
- 1 = indexation cyclique.

## Exécution

*table3* est semblable à *tablei*, sauf qu'il utilise l'interpolation cubique. (Nouveau dans la version 3.50 de Csound).

## Voir Aussi

*table*, *tablei*, *oscil1*, *oscil1i*, *osciln*

# tablecopy

tablecopy — Opcode de copie de table simple et rapide.

## Description

Opcode de copie de table simple et rapide.

## Syntaxe

**tablecopy** *kdft*, *ksft*

## Exécution

*kdft* -- Table de fonction destination.

*ksft* -- Numéro de la table de fonction source.

*tablecopy* -- Opcode de copie de table simple et rapide. Il prend la longueur de la table destination et lit à partir du début de la table source. Pour aller vite, il ne teste pas la longueur de la source - il copie quoi qu'il arrive - en mode « cyclique ». Ainsi, la table source peut-être lue plusieurs fois. Avec une table source de longueur 1, toutes les positions de la tables destination recevront son unique valeur.

*tablecopy* ne peut pas lire ou écrire le point de garde. Pour le lire, il faut utiliser *table*, avec *ndx* = la longueur de la table. De même, il faut utiliser une écriture de table pour l'écrire.

Pour écrire le point de garde avec la valeur de la position 0, utiliser *tablegpw*.

Cet opcode sert principalement à changer les tables de fonction rapidement dans une situation de temps réel.

## Voir Aussi

*tablegpw*, *tablemix*, *tableicopy*, *tableigpw*, *tableimix*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tablegpw

tablegpw — Ecrit le point de garde d'une table.

## Description

Ecrit le point de garde d'une table.

## Syntaxe

```
tablegpw kfn
```

## Exécution

*kfn* -- Numéro de la table.

*tablegpw* -- Pour écrire le point de garde d'une table, avec la valeur de la position 0. Ne fait rien si la table n'existe pas.

Peut être utile après avoir manipulé une table avec *tablemix* ou *tablecopy*.

## Voir Aussi

*tablecopy*, *tablemix*, *tableicopy*, *tableigpw*, *tableimix*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47



# tablei

tablei — Accède aux valeurs d'une table par indexation directe avec interpolation linéaire.

## Description

Accède aux valeurs d'une table par indexation directe avec interpolation linéaire.

## Syntaxe

```
ares tablei andx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ires tablei indx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
kres tablei kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction. *tablei* nécessite un point de garde.

*ixmode* (facultatif) -- type de l'index. La valeur par défaut est 0.

- 0 = index brut
- 1 = normalisé (de 0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est au centre, utiliser *taille\_table/2* (brut) ou 0.5 (normalisé). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = indexation normale (index < 0 traité comme index=0 ; index > *taille\_table* ramené à index=*taille\_table*)
- 1 = indexation cyclique.

## Exécution

*tablei* est une unité avec interpolation dans laquelle la partie fractionnaire de l'index est utilisée pour interpoler entre les entrées adjacentes de la table. La régularité apportée par l'interpolation se paie par une légère augmentation du temps d'exécution (voir aussi *oscili*, etc.), mais sinon les unités avec ou sans interpolation sont interchangeables. Noter que lorsque *tablei* utilise un index périodique dont la valeur modulo *n* est inférieure à la puissance de 2, longueur de la table, l'interpolation nécessite qu'il existe une (*n* + 1)ème valeur dans la table qui est une copie de la première valeur (voir l'*instruction f* de la partition).

## Voir Aussi

*table*, *table3*, *oscil1*, *oscil1i*, *osciln*

# tablecopy

tablecopy — Opcode de copie de table simple et rapide.

## Description

Opcode de copie de table simple et rapide.

## Syntaxe

```
tablecopy idft, isft
```

## Initialisation

*idft* -- Table de fonction destination.

*isft* -- Numéro de la table de fonction source.

## Exécution

*tablecopy* -- Opcode de copie de table simple et rapide. Il prend la longueur de la table destination et lit à partir du début de la table source. Pour aller vite, il ne teste pas la longueur de la source - il copie quoi-qu'il arrive - en mode « cyclique ». Ainsi, la table source peut-être lue plusieurs fois. Avec une table source de longueur 1, toutes les positions de la tables destination recevront son unique valeur.

*tablecopy* ne peut pas lire ou écrire le point de garde. Pour le lire, il faut utiliser *table*, avec *ndx* = la longueur de la table. De même, il faut utiliser une écriture de table pour l'écrire.

Pour écrire le point de garde avec la valeur de la position 0, utiliser *tablegpw*.

Cet opcode sert principalement à changer les tables de fonction rapidement dans une situation de temps réel.

## Voir Aussi

*tablecopy*, *tablegpw*, *tablemix*, *tableigpw*, *tableimix*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableigpw

tableigpw — Ecrit le point de garde d'une table.

## Description

Ecrit le point de garde d'une table.

## Syntaxe

```
tableigpw ifn
```

## Initialisation

*ifn* -- Numéro de la table.

## Exécution

*tableigpw* -- Pour écrire le point de garde d'une table, avec la valeur de la position 0. Ne fait rien si la table n'existe pas.

Peut être utile après avoir manipulé une table avec *tablemix* ou *tablecopy*.

## Voir Aussi

*tablecopy*, *tablegpw*, *tablemix*, *tableicopy*, *tableimix*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableikt

tableikt — Permet de contrôler au taux-k les numéros de table.

## Description

Contrôle des numéros de table au taux-k. La lecture dans la table se fait avec interpolation linéaire.

L'opcode standard *tablei* de Csound, bien que produisant un résultat au taux-k ou au taux-a, ne peut utiliser qu'une variable de taux-i pour choisir le numéro de la table. *tableikt* accepte un contrôle au taux-k aussi bien qu'au taux-i. Pour le reste, il est semblable à l'opcode original.

## Syntaxe

```
ares tableikt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

```
kres tableikt kndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ixmode* -- s'il vaut 0, *xndx* et *ixoff* couvrent toute la longueur de la table. S'il est différent de zéro, *xndx* et *ixoff* varient de 0 à 1. La valeur par défaut est 0.

*ixoff* -- s'il vaut 0, l'indice résultant est directement contrôlé par *xndx*, démarrant au début de la table. S'il est différent de zéro, l'indexation démarre à l'intérieur de la table. Sa valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0). La valeur par défaut est 0.

*iwrap* -- si *iwrap* = 0, *mode Limite* : lorsque l'indice résultant est inférieur à 0, l'indice final vaut 0. Un indice résultant dépassant la longueur de la table donne un indice final égal à la longueur de la table : les indices résultants trop grands se limitent à l'index supérieur de la table. Si *iwrap* est différent de 0, *mode Cyclique* : l'indice résultant est replié modulo la longueur de la table de façon à ce que tous les indices résultants tombent dans la table. Par exemple, dans une table de longueur 8, *xndx* = 5 et *ixoff* = 6 donnent un indice résultant de 11, qui se replie en un indice final de 3. La valeur par défaut est 0.

## Exécution

*kndx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*xndx* -- varie sur la longueur de la table (*ixmode* = 0) ou dans l'intervalle allant de 0 à 1 (*ixmode* différent de 0).

*kfn* -- Numéro de table. Doit être  $\geq 1$ . Les valeurs flottantes sont arrondies à un entier. Si un numéro de table n'indique pas une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur se produit et l'instrument est désactivé.



### Attention avec les numéros de table au taux-k

Au taux-k, si un numéro de table  $< 1$  est donné, ou si le numéro de table indique une table inexistante ou une table de longueur nulle (devant être chargée à partie d'un fichier ultérieurement), une erreur se produit et l'instrument est désactivé. *kfn* doit être initialisé au

taux approprié en utilisant *init*. Si l'on essaie de charger une valeur de taux-*i* dans *kfn*, il y aura une erreur.

## Voir Aussi

*tablekt*

## Crédits

Auteur: Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableimix

tableimix — Mélange deux tables.

## Description

Mélange deux tables.

## Syntaxe

```
tableimix idft, idoff, ilen, islft, isloff, islg, is2ft, is2off, is2g
```

## Initialisation

*idft* -- Table de fonction destination.

*idoff* -- Décalage de l'origine de l'écriture. Peut être négatif.

*ilen* -- Nombre d'opérations d'écriture à réaliser. Une valeur négative signifie écrire avec des indices descendants.

*islft*, *is2ft* -- Tables de fonction sources. Peuvent être identique à la table destination, si l'on fait attention au sens d'écriture lors de la copie des données.

*isloff*, *is2off* -- Décalages de l'origine de la lecture dans les tables sources.

*islg*, *is2g* -- Gains à appliquer lors de la lecture dans les tables source. Les résultats sont additionnés et la somme est écrite dans la table destination.

## Exécution

*tableimix* -- Cet opcode mélange deux tables, avec des gains séparés dans une table destination. L'écriture se fait sur *ilen* positions, habituellement en avançant dans la table si *ilen* est positif. S'il est négatif, l'écriture et la lecture se font avec des indices décroissants dans les tables. Cet option bi-directionnelle permet de déplacer facilement le contenu d'une table en lisant et en écrivant dans celle-ci avec un décalage différent.

Si *ilen* vaut 0, il n'y a pas d'écriture. Noter que la valeur entière interne de *ilen* est obtenue de la fonction *floor()* du C ANSI qui retourne l'entier négatif directement inférieur. Ainsi avec une valeur fractionnaire négative de *ilen* de -2.3 on aura une longueur interne de 3, et la copie commencera à partir des positions décalées et se fera sur deux positions vers la gauche.

L'indice résultant pour la lecture et l'écriture dans les tables est calculé à partir du décalage de l'origine pour chaque table auquel est additionnée la valeur de l'index, qui commence à 0 et augmente ou diminue d'un pas unité tout au long du mixage.

Ces indices résultants peuvent devenir très grands, car il n'y a aucune restriction pour le décalage ou *ilen*. Cependant l'indice résultant pour chaque table subit un ET logique avec un masque de longueur (tel que 0000 0111 pour une table de longueur 8) pour former l'indice final qui sera utilisé pour la lecture ou l'écriture. Ainsi il ne peut y avoir aucune lecture ou écriture en dehors des tables. C'est la même chose que le mode « wrap » (cyclique) dans la lecture et l'écriture de table. Ces opcodes ne lisent pas ou n'écrivent pas le point de garde. Si une table a été réécrite par l'un de ceux-ci et si elle a un point de garde sensé contenir la même valeur que la position 0, il faut ensuite appeler *tableigpw*.

Les indices et les décalages sont exprimés en pas de table - ils ne sont pas normalisés entre 0 et 1. Ainsi pour une table de longueur 256, *ilen* doit être fixé à 256 si toute la table doit être lue ou écrite.

Il n'est pas nécessaire que les tables soient de même longueur - le parcours cyclique se fait individuellement pour chaque table.

## Voir Aussi

*tablecopy*, *tablegpw*, *tablemix*, *tableicopy*, *tableigpw*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableiw

tableiw — Change le contenu de tables de fonction existantes.

## Description

Cet opcode opère sur des tables de fonction existantes en changeant leur contenu. *tableiw* est utilisé quand toutes les entrées sont des variables de taux-i ou des constantes et que l'on veut seulement l'exécuter à l'initialisation de l'instrument. Les combinaisons valides des types de variable sont indiquées par la première lettre des noms de variable.

## Syntaxe

```
tableiw isig, indx, ifn [, ixmode] [, ixoff] [, iwgmode]
```

## Initialisation

*isig* -- Valeur d'entrée à écrire dans la table.

*indx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*ifn* -- Numéro de la table. Doit être  $\geq 1$ . Les nombres flottants sont arrondis à l'entier inférieur. Si un numéro de table ne pointe pas vers une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur est générée et l'instrument est désactivé.

*ixmode* (facultatif, 0 par défaut) -- mode d'indexation.

- 0 = *indx* et *ixoff* sont compris entre 0 et la longueur de la table.
- différent de 0 = *indx* et *ixoff* sont compris entre 0 et 1.

*ixoff* (facultatif, 0 par défaut) -- décalage de l'index.

- 0 = l'indice résultant est contrôlé directement par *indx*, l'indexation commençant depuis le début de la table.
- Différent de 0 = l'indexation démarre dans la table. La valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0).

*iwgmode* (facultatif, 0 par défaut) -- mode cyclique et point de garde.

- 0 = mode limite.
- 1 = mode cyclique.
- 2 = mode point de garde.



## Exécution

### Mode limite (0)

Limite l'indice résultant ( $indx + ixoff$ ) entre 0 et le point de garde. Pour une table de longueur 5, cela signifie que les positions allant de 0 à 3 et la position 4 (le point de garde) peuvent être écrites. Un indice résultant négatif provoque l'écriture en position 0.

### Mode cyclique (1)

Parcours cyclique de l'indice résultant dans les positions 0 à E, où E vaut soit la longueur de la table moins un, soit le facteur de 2 qui est égal à la longueur de la table moins un. Par exemple, un parcours cyclique entre 0 et 3, si bien que l'indice 6 signifie une écriture dans la position 2.

### Mode point de garde (2)

Le point de garde est écrit en même temps que la position 0 avec la même valeur.

Facilite l'écriture dans des tables prévues pour être lues avec interpolation pour produire des formes d'onde cycliques sans discontinuité. De plus, avant son utilisation, l'indice résultant est augmenté de la moitié de la distance entre une position et la suivante, avant d'être arrondi à l'adresse entière inférieure d'une position dans la table.

Normalement ( $igwmode = 0$  ou  $1$ ), pour une table de longueur 5, qui comprend les positions 0 à 3 en partie principale et la position 4 comme point de garde, un indice résultant compris entre 0 et 0.999 provoquera une écriture dans la position 0. ("0.999" signifie juste inférieur à 1.0), entre 1.0 et 1.999, l'écriture se fera dans la position 1, etc. La même interprétation a lieu pour les indices résultants compris entre 0 et 4.999 ( $igwmode = 0$ ) ou 3.999 ( $igwmode = 1$ ).  $igwmode = 0$  permet l'écriture dans les positions 0 à 4, avec la possibilité d'avoir dans le point de garde (4) une valeur différente de celle de la position 0.

Avec une table de longueur 5 et  $igwmode = 2$ , quand l'indice résultant est compris entre 0 et 0.499, l'écriture se fera dans les positions 0 et 4. S'il est compris entre 0.5 et 1.499, l'écriture se fera dans la position 1, etc. S'il est compris entre 3.5 et 4.0, l'écriture se fera également dans les positions 0 et 4.

Ainsi, l'écriture s'approche le plus possible des résultats de la lecture avec interpolation. Le mode point de garde ne doit être utilisé qu'avec des tables qui ont un point de garde.

Le mode point de garde se fait en ajoutant 0.5 à l'indice résultant, en l'arrondissant à l'entier inférieur le plus proche, puis en le réduisant modulo le facteur de deux égal à la longueur de la table moins un, enfin en écrivant dans la table (positions 0 à 3 dans notre exemple) et dans le point de garde si l'indice vaut 0.

## Voir Aussi

*tablew*, *tablewkt*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

Mise à jour en août 2002, grâce à Abram Hindle qui a indiqué le syntaxe correcte.

# tablekt

tablekt — Permet de contrôler au taux-k les numéros de table.

## Description

Contrôle des numéros de table au taux-k.

L'opcode standard *table* de Csound, bien que produisant un résultat au taux-k ou au taux-a, ne peut utiliser qu'une variable de taux-i pour choisir le numéro de la table. *tablekt* accepte un contrôle au taux-k aussi bien qu'au taux-i. Pour le reste, il est semblable à l'opcode original.

## Syntaxe

```
ares tablekt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

```
kres tablekt kndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ixmode* -- s'il vaut 0, *xndx* et *ixoff* couvrent toute la longueur de la table. S'il est différent de zéro, *xndx* et *ixoff* varient de 0 à 1. La valeur par défaut est 0.

*ixoff* -- s'il vaut 0, l'indice résultant est directement contrôlé par *xndx*, démarrant au début de la table. S'il est différent de zéro, l'indexation démarre à l'intérieur de la table. Sa valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0). La valeur par défaut est 0.

*iwrap* -- si *iwrap* = 0, *mode Limite* : lorsque l'indice résultant est inférieur à 0, l'indice final vaut 0. Un indice résultant dépassant la longueur de la table donne un indice final égal à la longueur de la table : les indices résultants trop grands se limitent à l'index supérieur de la table. Si *iwrap* est différent de 0, *mode Cyclique* : l'indice résultant est replié modulo la longueur de la table de façon à ce que tous les indices résultants tombent dans la table. Par exemple, dans une table de longueur 8, *xndx* = 5 et *ixoff* = 6 donnent un indice résultant de 11, qui se replie en un indice final de 3. La valeur par défaut est 0.

## Exécution

*kndx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*xndx* -- varie sur la longueur de la table (*ixmode* = 0) ou dans l'intervalle allant de 0 à 1 (*ixmode* différent de 0).

*kfn* -- Numéro de table. Doit être  $\geq 1$ . Les valeurs flottantes sont arrondies à un entier. Si un numéro de table n'indique pas une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur se produit et l'instrument est désactivé.



### Attention avec les numéros de table au taux-k

Au taux-k, si un numéro de table  $< 1$  est donné, ou si le numéro de table indique une table inexistante ou une table de longueur nulle (devant être chargée à partie d'un fichier ultérieurement), une erreur se produit et l'instrument est désactivé. *kfn* doit être initialisé au

taux approprié en utilisant *init*. Si l'on essaie de charger une valeur de taux-*i* dans *kfn*, il y aura une erreur.

## Voir Aussi

*tableikt*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tablemix

tablemix — Mélange deux tables.

## Description

Mélange deux tables.

## Syntaxe

**tablemix** *kdft*, *kdoff*, *klen*, *ks1ft*, *ks1off*, *ks1g*, *ks2ft*, *ks2off*, *ks2g*

## Exécution

*kdft* -- Table de fonction destination.

*kdoff* -- Décalage de l'origine de l'écriture. Peut être négatif.

*klen* -- Nombre d'opérations d'écriture à réaliser. Une valeur négative signifie écrire avec des indices descendants.

*ks1ft*, *ks2ft* -- Tables de fonction sources. Peuvent être identique à la table destination, si l'on fait attention au sens d'écriture lors de la copie des données.

*ks1off*, *ks2off* -- Décalages de l'origine de la lecture dans les tables sources.

*ks1g*, *ks2g* -- Gains à appliquer lors de la lecture dans les tables source. Les résultats sont additionnés et la somme est écrite dans la table destination.

*tablemix* -- Cet opcode mélange deux tables, avec des gains séparés dans une table destination. L'écriture se fait sur *klen* positions, habituellement en avançant dans la table si *klen* est positif. S'il est négatif, l'écriture et la lecture se font avec des indices décroissants dans les tables. Cet option bi-directionnelle permet de déplacer facilement le contenu d'une table en lisant et en écrivant dans celle-ci avec un décalage différent.

Si *klen* vaut 0, il n'y a pas d'écriture. Noter que la valeur entière interne de *klen* est obtenue de la fonction *floor()* du C ANSI qui retourne l'entier négatif directement inférieur. Ainsi avec une valeur fractionnaire négative de *klen* de -2.3 on aura une longueur interne de 3, et la copie commencera à partir des positions décalées et se fera sur deux positions vers la gauche.

L'indice résultant pour la lecture et l'écriture dans les tables est calculé à partir du décalage de l'origine pour chaque table auquel est additionnée la valeur de l'index, qui commence à 0 et augmente ou diminue d'un pas unité tout au long du mixage.

Ces indices résultants peuvent devenir très grands, car il n'y a aucune restriction pour le décalage ou *klen*. Cependant l'indice résultant pour chaque table subit un ET logique avec un masque de longueur (tel que 0000 0111 pour une table de longueur 8) pour former l'indice final qui sera utilisé pour la lecture ou l'écriture. Ainsi il ne peut y avoir aucune lecture ou écriture en dehors des tables. C'est la même chose que le mode « wrap » (cyclique) dans la lecture et l'écriture de table. Ces opcodes ne lisent pas ou n'écrivent pas le point de garde. Si une table a été réécrite par l'un de ceux-ci et si elle a un point de garde sensé contenir la même valeur que la position 0, il faut ensuite appeler *tablegpw* afterwards.

Les indices et les décalages sont exprimés en pas de table - ils ne sont pas normalisés entre 0 et 1. Ainsi pour une table de longueur 256, *klen* doit être fixé à 256 si toute la table doit être lue ou écrite.

Il n'est pas nécessaire que les tables soient de même longueur - le parcours cyclique se fait individuellement pour chaque table.

## Voir Aussi

*tablecopy, tablegpw, tableicopy, tableigpw, tableimix*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableng

tableng — Interroge une table de fonction sur sa longueur.

## Description

Interroge une table de fonction sur sa longueur.

## Syntaxe

```
ires tableng ifn
```

```
kres tableng kfn
```

## Initialisation

*ifn* -- Numéro de la table à interroger.

## Exécution

*kfn* -- Numéro de la table à interroger.

*tableng* retourne la longueur de la table spécifiée. Ce sera une puissance de deux dans la plupart des cas. N'indique pas si une table a ou non un point de garde. Il semble que cette information ne soit pas disponible dans la structure de données de la table. Si la table spécifiée n'est pas trouvée, retourne 0.

Peut-être utile pour configurer le code d'opérations de manipulation de table, comme *tablemix* et *table-copy*.

## Exemples

Voici un exemple de l'opcode *tableng*. Il utilise le fichier *tableng.csd* [examples/tableng.csd].

### Exemple 518. Exemple de l'opcode *tableng*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o tableng.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Let's look at Table #1.
  ifn = 1
  ilen tableng ifn

  print ilen
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

La table a une longueur de 16384 échantillons. La sortie comprendra donc une ligne comme celle-ci :

```
instr 1:  ilen = 16384.000
```

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Exemple écrit par Kevin Conder.

# tablera

tablera — Reads tables in sequential locations.

## Description

These opcode reads tables in sequential locations to an a-rate variable. Some thought is required before using it. It has at least two major, and quite different, applications which are discussed below.

## Syntax

```
ares tablera kfn, kstart, koff
```

## Performance

*ares* -- a-rate destination for reading *ksmps* values from a table.

*kfn* -- i- or k-rate number of the table to read or write.

*kstart* -- Where in table to read or write.

*koff* -- i- or k-rate offset into table. Range unlimited - see explanation at end of this section.

In one application, *tablera* is intended to be used in pair with *tablewa*, or with several *tablera* opcodes before a *tablewa* -- all sharing the same *kstart* variable.

These read from and write to sequential locations in a table at audio rates, with *ksmps* floats being written and read each cycle.

*tablera* starts reading from location *kstart*. *tablewa* starts writing to location *kstart*, and then writes to *kstart* with the number of the location one more than the one it last wrote. (Note that for *tablewa*, *kstart* is both an input and output variable.) If the writing index reaches the end of the table, then no further writing occurs and zero is written to *kstart*.

For instance, if the table's length was 16 (locations 0 to 15), and *ksmps* was 5. Then the following steps would occur with repetitive runs of the *tablewa* opcode, assuming that *kstart* started at 0.

Run Number	Initial kstart	Final kstart	Locations Written
1	0	5	0 1 2 3 4
2	5	10	5 6 7 8 9
3	10	15	10 11 12 13 14
4	15	0	15

This is to facilitate processing table data using standard a-rate orchestra code between the *tablera* and *tablewa* opcodes. They allow all Csound k-rate operators to be used (with caution) on a-rate variables - something that would only be possible otherwise by *ksmps* = 1, *downsamp* and *upsamp*.



### Several cautions



- The k-rate code in the processing loop is really running at a-rate, so time dependent functions like *port* and *oscil* work faster than normal - their code is expecting to be running at k-rate.
- This system will produce undesirable results unless the *ksmps* fits within the table length. For instance a table of length 16 will accommodate 1 to 16 samples, so this example will work with *ksmps* = 1 to 16.

Both these opcodes generate an error and deactivate the instrument if a table with length  $< ksmpls$  is selected. Likewise an error occurs if *kstart* is below 0 or greater than the highest entry in the table - if *kstart* = table length.

- *kstart* is intended to contain integer values between 0 and (table length - 1). Fractional values above this should not affect operation but do not achieve anything useful.
- These opcodes are not interpolating, and the *kstart* and *koff* parameters always have a range of 0 to (table length - 1) - not 0 to 1 as is available in other table read/write opcodes. *koff* can be outside this range but it is wrapped around by the final AND operation.
- These opcodes are permanently in wrap mode. When *koff* is 0, no wrapping needs to occur, since the *kstart*++ index will always be within the table's normal range. *koff* not equal to 0 can lead to wrapping.
- The offset does not affect the number of read/write cycles performed, or the value written to *kstart* by *tablewa*.
- These opcodes cannot read or write the guardpoint. Use *tablegpw* to write the guardpoint after manipulations have been done with *tablewa*.

## Examples

```
kstart    =      0
lab1:
  atemp    tablewa ktabsource, kstart, 0 ; Read 5 values from table into an
      ; a-rate variable.

  atemp    =      log(atemp) ; Process the values using a-rate
      ; code.

  kstart    tablewa ktabdest, atemp, 0 ; Write it back to the table
if ktemp 0 goto lab1 ; Loop until all table locations
      ; have been processed.
```

The above example shows a processing loop, which runs every k-cycle, reading each location in the table *ktabsource*, and writing the log of those values into the same locations of table *ktabdest*.

This enables whole tables, parts of tables (with offsets and different control loops) and data from several tables at once to be manipulated with a-rate code and written back to another (or to the same) table. This is a bit of a fudge, but it is faster than doing it with k-rate table read and write code.

Another application is:

```
kzero = 0
kloop = 0

kzero tablewa 23, asignal, 0 ; ksmps a-rate samples written
    ; into locations 0 to (ksmps -1) of table 23.

labl: ktemp table kloop, 23 ; Start a loop which runs ksmps times,
    ; in which each cycle processes one of
    [ Some code to manipulate ] ; table 23's values with k-rate orchestra
    [ the value of ktemp. ] ; code.

    tablew ktemp, kloop, 23 ; Write the processed value to the table.

kloop = kloop + 1 ; Increment the kloop, which is both the
    ; pointer into the table and the loop
if kloop < ksmps goto labl ; counter. Keep looping until all values
    ; in the table have been processed.

asignal tablewa 23, 0, 0 ; Copy the table contents back
    ; to an a-rate variable.
```

*koff* -- This is an offset which is added to the sum of *kstart* and the internal index variable which steps through the table. The result is then ANDed with the lengthmask (000 0111 for a table of length 8 - or 9 with guardpoint) and that final index is used to read or write to the table. *koff* can be any value. It is converted into a long using the ANSI floor() function so that -4.3 becomes -5. This is what we would want when using offsets which range above and below zero.

Ideally this would be an optional variable, defaulting to 0, however with the existing Csound orchestra read code, such default parameters must be init time only. We want k-rate here, so we cannot have a default.

## See Also

*tablewa*

# tableseg

**tableseg** — Creates a new function table by making linear segments between values in stored function tables.

## Description

*tableseg* is like *linseg* but interpolate between values in a stored function tables. The result is a new function table passed internally to any following *vpvoc* which occurs before a subsequent *tableseg* (much like *lpread/lpreson* pairs work). The uses of these are described below under *vpvoc*.

## Syntax

```
tableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

## Initialization

*ifn1*, *ifn2*, *ifn3*, etc. -- function table numbers. *ifn1*, *ifn2*, and so on, must be the same size.

*idur1*, *idur2*, etc. -- durations during which interpolation from one table to the next will take place.

## See Also

*pvbufread*, *pvcross*, *pvinterp*, *pvread*, *tablexseg*

## Credits

Author: Richard Karpen  
Seattle, Wash  
1997

New in version 3.44

# tablew

tablew — Change le contenu de tables de fonction existantes.

## Description

Cet opcode opère sur des tables de fonction existantes en changeant leur contenu. *tablew* sert à l'écriture au taux-k ou au taux-a, le numéro de table étant spécifié à durant l'initialisation. Les combinaisons valides des types de variable sont indiquées par la première lettre des noms de variable.

## Syntaxe

```
tablew asig, andx, ifn [, ixmode] [, ixoff] [, iwemode]
```

```
tablew isig, indx, ifn [, ixmode] [, ixoff] [, iwemode]
```

```
tablew ksig, kndx, ifn [, ixmode] [, ixoff] [, iwemode]
```

## Initialisation

*asig, isig, ksig* -- La valeur à écrire dans la table.

*andx, indx, kndx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*ifn* -- Numéro de la table. Doit être  $\geq 1$ . Les nombres flottants sont arrondis à l'entier inférieur. Si un numéro de table ne pointe pas vers une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur est générée et l'instrument est désactivé.

*ixmode* (facultatif, 0 par défaut) -- mode d'indexation.

- 0 = *xndx* et *ixoff* sont compris entre 0 et la longueur de la table.
- différent de 0 = *xndx* et *ixoff* sont compris entre 0 et 1.

*ixoff* (facultatif, 0 par défaut) -- décalage de l'index.

- 0 = l'indice résultant est contrôlé directement par *xndx*, l'indexation commençant depuis le début de la table.
- Différent de 0 = l'indexation démarre dans la table. La valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0).

*iwemode* (facultatif, 0 par défaut) -- mode cyclique et point de garde.

- 0 = mode limite.
- 1 = mode cyclique.

- 2 = mode point de garde.

## Exécution

### Mode limite (0)

Limite l'indice résultant ( $xndx + ixoff$ ) entre 0 et le point de garde. Pour une table de longueur 5, cela signifie que les positions allant de 0 à 3 et la position 4 (le point de garde) peuvent être écrites. Un indice résultant négatif provoque l'écriture en position 0.

### Mode cyclique (1)

Parcours cyclique de l'indice résultant dans les positions 0 à E, où E vaut soit la longueur de la table moins un, soit le facteur de 2 qui est égal à la longueur de la table moins un. Par exemple, un parcours cyclique entre 0 et 3, si bien que l'indice 6 signifie une écriture dans la position 2.

### Mode point de garde (2)

Le point de garde est écrit en même temps que la position 0 avec la même valeur.

Facilite l'écriture dans des tables prévues pour être lues avec interpolation pour produire des formes d'onde cycliques sans discontinuité. De plus, avant son utilisation, l'indice résultant est augmenté de la moitié de la distance entre une position et la suivante, avant d'être arrondi à l'adresse entière inférieure d'une position dans la table.

Normalement ( $igwmode = 0$  ou  $1$ ), pour une table de longueur 5, qui comprend les positions 0 à 3 en partie principale et la position 4 comme point de garde, un indice résultant compris entre 0 et 0.999 provoquera une écriture dans la position 0. ("0.999" signifie juste inférieur à 1.0), entre 1.0 et 1.999, l'écriture se fera dans la position 1, etc. La même interprétation a lieu pour les indices résultants compris entre 0 et 4.999 ( $igwmode = 0$ ) ou 3.999 ( $igwmode = 1$ ).  $igwmode = 0$  permet l'écriture dans les positions 0 à 4, avec la possibilité d'avoir dans le point de garde (4) une valeur différente de celle de la position 0.

Avec une table de longueur 5 et  $igwmode = 2$ , quand l'indice résultant est compris entre 0 et 0.499, l'écriture se fera dans les positions 0 et 4. S'il est compris entre 0.5 et 1.499, l'écriture se fera dans la position 1, etc. S'il est compris entre 3.5 et 4.0, l'écriture se fera également dans les positions 0 et 4.

Ainsi, l'écriture s'approche le plus possible des résultats de la lecture avec interpolation. Le mode point de garde ne doit être utilisé qu'avec des tables qui ont un point de garde.

Le mode point de garde se fait en ajoutant 0.5 à l'indice résultant, en l'arrondissant à l'entier inférieur le plus proche, puis en le réduisant modulo le facteur de deux égal à la longueur de la table moins un, enfin en écrivant dans la table (positions 0 à 3 dans notre exemple) et dans le point de garde si l'indice vaut 0.

*tablew* ne retourne pas de valeur. Les trois derniers paramètres sont facultatifs et valent 0 par défaut.

## Avertissement pour les numéros de table de taux-k

Au taux-k ou au taux-a, si l'on donne un numéro de table  $< 1$ , ou si le numéro de table pointe vers une table inexistante ou vers une table de longueur nulle (qui doit être chargée depuis un fichier ultérieurement), une erreur est générée et l'instrument est désactivé. Il faut initialiser *kfn* et *afn* au taux approprié en utilisant *init*. Si l'on essaie de mettre une valeur de taux-i dans *kfn* ou dans *afn* une erreur est générée.

## Voir Aussi

*tableiw*, *tablewkt*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# tablewa

tablewa — Writes tables in sequential locations.

## Description

This opcode writes to a table in sequential locations to and from an a-rate variable. Some thought is required before using it. It has at least two major, and quite different, applications which are discussed below.

## Syntax

```
kstart tablewa kfn, asig, koff
```

## Performance

*kstart* -- Where in table to read or write.

*kfn* -- i- or k-rate number of the table to read or write.

*asig* -- a-rate signal to read from when writing to the table.

*koff* -- i- or k-rate offset into table. Range unlimited - see explanation at end of this section.

In one application, it is intended to be used with one or with several *tablera* opcodes before a *tablewa* -- all sharing the same *kstart* variable.

These read from and write to sequential locations in a table at audio rates, with *ksmps* floats being written and read each cycle.

*tablera* starts reading from location *kstart*. *tablewa* starts writing to location *kstart*, and then writes to *kstart* with the number of the location one more than the one it last wrote. (Note that for *tablewa*, *kstart* is both an input and output variable.) If the writing index reaches the end of the table, then no further writing occurs and zero is written to *kstart*.

For instance, if the table's length was 16 (locations 0 to 15), and *ksmps* was 5. Then the following steps would occur with repetitive runs of the *tablewa* opcode, assuming that *kstart* started at 0.

Run Number	Initial kstart	Final kstart	Locations Written
1	0	5	0 1 2 3 4
2	5	10	5 6 7 8 9
3	10	15	10 11 12 13 14
4	15	0	15

This is to facilitate processing table data using standard a-rate orchestra code between the *tablera* and *tablewa* opcodes. They allow all Csound k-rate operators to be used (with caution) on a-rate variables - something that would only be possible otherwise by *ksmps* = 1, *downsamp* and *upsamp*.



### Several cautions

- The k-rate code in the processing loop is really running at a-rate, so time dependent functions like *port* and *oscil* work faster than normal - their code is expecting to be running at k-rate.
- This system will produce undesirable results unless the *ksmps* fits within the table length. For instance a table of length 16 will accommodate 1 to 16 samples, so this example will work with *ksmps* = 1 to 16.

Both these opcodes generate an error and deactivate the instrument if a table with length  $< ksmpls$  is selected. Likewise an error occurs if *kstart* is below 0 or greater than the highest entry in the table - if *kstart* = table length.

- *kstart* is intended to contain integer values between 0 and (table length - 1). Fractional values above this should not affect operation but do not achieve anything useful.
- These opcodes are not interpolating, and the *kstart* and *koff* parameters always have a range of 0 to (table length - 1) - not 0 to 1 as is available in other table read/write opcodes. *koff* can be outside this range but it is wrapped around by the final AND operation.
- These opcodes are permanently in wrap mode. When *koff* is 0, no wrapping needs to occur, since the *kstart*++ index will always be within the table's normal range. *koff* not equal to 0 can lead to wrapping.
- The offset does not affect the number of read/write cycles performed, or the value written to *kstart* by *tablewa*.
- These opcodes cannot read or write the guardpoint. Use *tablegpw* to write the guardpoint after manipulations have been done with *tablewa*.

## Examples

```
kstart    =      0
lab1:
  atemp    tablera ktabsource, kstart, 0 ; Read 5 values from table into an
      ; a-rate variable.

  atemp    =      log(atemp) ; Process the values using a-rate
      ; code.

  kstart    tablewa ktabdest, atemp, 0 ; Write it back to the table
if ktemp 0 goto lab1 ; Loop until all table locations
      ; have been processed.
```

The above example shows a processing loop, which runs every k-cycle, reading each location in the table *ktabsource*, and writing the log of those values into the same locations of table *ktabdest*.

This enables whole tables, parts of tables (with offsets and different control loops) and data from several tables at once to be manipulated with a-rate code and written back to another (or to the same) table. This is a bit of a fudge, but it is faster than doing it with k-rate table read and write code.



Another application is:

```
kzero = 0
kloop = 0

kzero tablewa 23, asignal, 0 ; ksmps a-rate samples written
    ; into locations 0 to (ksmps -1) of table 23.

lab1: ktemp table kloop, 23 ; Start a loop which runs ksmps times,
    ; in which each cycle processes one of
    [ Some code to manipulate ] ; table 23's values with k-rate orchestra
    [ the value of ktemp. ] ; code.

    tablew ktemp, kloop, 23 ; Write the processed value to the table.

kloop = kloop + 1 ; Increment the kloop, which is both the
    ; pointer into the table and the loop
if kloop < ksmps goto lab1 ; counter. Keep looping until all values
    ; in the table have been processed.

asignal tablewa 23, 0, 0 ; Copy the table contents back
    ; to an a-rate variable.
```

*koff* -- This is an offset which is added to the sum of *kstart* and the internal index variable which steps through the table. The result is then ANDed with the lengthmask (000 0111 for a table of length 8 - or 9 with guardpoint) and that final index is used to read or write to the table. *koff* can be any value. It is converted into a long using the ANSI floor() function so that -4.3 becomes -5. This is what we would want when using offsets which range above and below zero.

Ideally this would be an optional variable, defaulting to 0, however with the existing Csound orchestra read code, such default parameters must be init time only. We want k-rate here, so we cannot have a default.

## Credits

Author: Robin Whittle  
Australia

# tablewkt

tablewkt — Change the contents of existing function tables.

## Description

This opcode operates on existing function tables, changing their contents. *tablewkt* uses a k-rate variable for selecting the table number. The valid combinations of variable types are shown by the first letter of the variable names.

## Syntax

```
tablewkt asig, andx, kfn [, ixmode] [, ixoff] [, iwgmode]
```

```
tablewkt ksig, kndx, kfn [, ixmode] [, ixoff] [, iwgmode]
```

## Initialization

*asig, ksig* -- The value to be written into the table.

*andx, kndx* -- Index into table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0)

*kfn* -- Table number. Must be >= 1. Floats are rounded down to an integer. If a table number does not point to a valid table, or the table has not yet been loaded (*GEN01*) then an error will result and the instrument will be de-activated.

*ixmode* -- index mode. Default is zero.

- 0 = *xndx* and *ixoff* ranges match the length of the table.
- Not equal to 0 = *xndx* and *ixoff* have a 0 to 1 range.

*ixoff* -- index offset. Default is 0.

- 0 = Total index is controlled directly by *xndx*, i.e. the indexing starts from the start of the table.
- Not equal to 0 = Start indexing from somewhere else in the table. Value must be positive and less than the table length (*ixmode* = 0) or less than 1 (*ixmode* != 0).

*iwgmode* -- table writing mode. Default is 0.

- 0 = Limit mode.
- 1 = Wrap mode.
- 2 = Guardpoint mode.

## Performance

### Limit mode (0)

Limit the total index ( $ndx + ixoff$ ) to between 0 and the guard point. For a table of length 5, this means that locations 0 to 3 and location 4 (the guard point) can be written. A negative total index writes to location 0.

### Wrap mode (1)

Wrap total index value into locations 0 to E, where E is one less than either the table length or the factor of 2 number which is one less than the table length. For example, wrap into a 0 to 3 range - so that total index 6 writes to location 2.

### Guardpoint mode (2)

The guardpoint is written at the same time as location 0 is written - with the same value.

This facilitates writing to tables which are intended to be read with interpolation for producing smooth cyclic waveforms. In addition, before it is used, the total index is incremented by half the range between one location and the next, before being rounded down to the integer address of a table location.

Normally ( $igwmode = 0$  or  $1$ ) for a table of length 5 - which has locations 0 to 3 as the main table and location 4 as the guard point, a total index in the range of 0 to 0.999 will write to location 0. ("0.999" means just less than 1.0.) 1.0 to 1.999 will write to location 1 etc. A similar pattern holds for all total indexes 0 to 4.999 ( $igwmode = 0$ ) or to 3.999 ( $igwmode = 1$ ).  $igwmode = 0$  enables locations 0 to 4 to be written - with the guardpoint (4) being written with a potentially different value from location 0.

With a table of length 5 and the  $igwmode = 2$ , then when the total index is in the range 0 to 0.499, it will write to locations 0 and 4. Range 0.5 to 1.499 will write to location 1 etc. 3.5 to 4.0 will also write to locations 0 and 4.

This way, the writing operation most closely approximates the results of interpolated reading. Guard point mode should only be used with tables that have a guardpoint.

Guardpoint mode is accomplished by adding 0.5 to the total index, rounding to the next lowest integer, wrapping it modulo the factor of two which is one less than the table length, writing the table (locations 0 to 3 in our example) and then writing to the guard point if index = 0.

## Caution with k-rate table numbers

At k-rate or a-rate, if a table number of  $< 1$  is given, or the table number points to a non-existent table, or to one which has a length of 0 (it is to be loaded from a file later) then an error will result and the instrument will be deactivated.  $kfn$  and  $afn$  must be initialized at the appropriate rate using *init*. Attempting to load an i-rate value into  $kfn$  or  $afn$  will result in an error.

## See Also

*tableiw*, *tablew*

## Credits

Author: Robin Whittle  
Australia  
May 1997



# tablexkt

tablexkt — Lit des tables de fonction avec interpolation linéaire, cubique ou sinc.

## Description

Lit des tables de fonction avec interpolation linéaire, cubique ou sinc.

## Syntaxe

```
ares tablexkt xndx, kfn, kwarp, iwsiz [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*iwsiz* -- Ce paramètre contrôle le type d'interpolation à utiliser :

- 2 : Interpolation linéaire. C'est la qualité la plus faible, mais aussi le mode le plus rapide.
- 4 : Interpolation cubique. Qualité légèrement meilleure qu'avec *iwsiz* = 2, au prix d'un traitement moins rapide.
- 8 et au-dessus (jusqu'à 1024) : interpolation sinc avec une taille de fenêtre égale à *iwsiz* (doit être un multiple entier de 4). Meilleure qualité que l'interpolation linéaire ou cubique, mais très lent. Lorsque l'on transpose vers le haut, on peut utiliser une valeur de *kwarp* supérieure à 1 pour un meilleur lissage (c'est encore plus lent).

*ixmode* (facultatif) -- mode d'indexation. La valeur par défaut est 0.

- 0 : indices bruts
- valeur différente de 0 : normalisé (0 à 1)



## Notes

Si l'on utilise *tablexkt* pour reproduire des échantillons avec boucle (par exemple avec un indice de table généré par *lphasor*), il faut qu'il y ait au moins *iwsiz* / 2 échantillons après la fin de la boucle pour l'interpolation, sinon il pourra y avoir des clics audibles (il doit y avoir aussi au moins *iwsiz* / 2 échantillons avant le début de la boucle).

*ixoff* (facultatif) -- valeur de décalage de l'indice. Pour une table dont l'origine est au centre, il faut utiliser *taille\_table* / 2 (indices bruts) ou 0.5 (indices normalisés). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur de parcours cyclique des indices. La valeur par défaut est 0.

- 0 : Pas de lecture cyclique (les indices < 0 sont ramenés à 0 ; les indices >= à la taille de la table (ou 1.0 en mode normalisé) restent bloqués sur le point de garde).
- valeur différente de 0 : l'indice est replié dans l'intervalle autorisé (sans inclure le point de garde dans

ce cas).



## Note

*iwrap* s'applique aussi aux échantillons supplémentaires pour l'interpolation.

## Exécution

*ares* -- sortie audio.

*xndx* -- index de la table.

*kfn* -- numéro de la table de fonction.

*kwarp* -- s'il est supérieur à 1, on utilise la fonction  $\sin(x / \text{kwarp}) / x$  pour l'interpolation sinc au lieu de la fonction par défaut  $\sin(x) / x$ . C'est utile pour lisser lorsque l'on transpose vers le haut (*kwarp* doit être fixé au facteur de transposition dans ce cas, par exemple 2.0 pour une octave), cependant le rendu peut-être jusqu'à deux fois plus lent. De plus, *iwsz* doit valoir au moins  $\text{kwarp} * 8$ . Cette possibilité est expérimentale et pourra être améliorée à la fois en termes de vitesse et de qualité dans de nouvelles versions.



## Note

*kwarp* n'a aucun effet s'il est inférieur ou égal à 1, ou si l'interpolation linéaire ou cubique est utilisée.

## Exemples

Voici un exemple de l'opcode *tablexkt*. Il utilise le fichier *tablexkt.csd* [examples/tablexkt.csd].

### Exemple 519. Exemple de l'opcode *tablexkt*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tablexkt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Jonathan Murphy

sr      = 44100
ksmps   = 10
nchnls  = 1

instr 1

ifn      = 1      ; query fl as to number of samples
ilen     = nsamp(ifn)

itrns    = 4      ; transpose up 4 octaves
ilps     = 16     ; allow iwsz/2 samples at start
ilpe     = ilen - 16 ; and at end
```

```
imode      = 3 ; loop forwards and backwards
istrt      = 16 ; start 16 samples into loop

alphs      lphasor  itrns, ilps, ilpe, imode, istrt
           ; use lphasor as index
andx       = alphs

kfn         = 1 ; read f1
kwarp       = 4 ; anti-aliasing, should be same value as itrns above
iwsiz      = 32 ; iwsiz must be at least 8 * kwarp

atab       tablexkt andx, kfn, kwarp, iwsiz

atab       = atab * 10000

           out      atab

        endin

</CsInstruments>
<CsScore>
f 1 0 262144 1 "beats.wav" 0 4 1
il 0 60
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Istvan Varga  
Janvier 2002  
Exemple par Jonathan Murphy 2006

Nouveau dans la version 4.18

# tablexseg

*tablexseg* — Creates a new function table by making exponential segments between values in stored function tables.

## Description

*tablexseg* is like *expseg* but interpolate between values in a stored function tables. The result is a new function table passed internally to any following *vpvoc* which occurs before a subsequent *tablexseg* (much like *lpread/lpreson* pairs work). The uses of these are described below under *vpvoc*.

## Syntax

```
tablexseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

## Initialization

*ifn1*, *ifn2*, *ifn3*, etc. -- function table numbers. *ifn1*, *ifn2*, and so on, must be the same size.

*idur1*, *idur2*, etc. -- durations during which interpolation from one table to the next will take place.

## See Also

*pvbufread*, *pvcross*, *pvinterp*, *pvread*, *tableseg*

## Credits

Author: Richard Karpen  
Seattle, WA USA  
1997



# tabmorph

tabmorph — Allow morphing between a set of tables.

## Description

*tabmorph* allows morphing between a set of tables of the same size, by means of a weighted average between two currently selected tables.

## Syntax

```
kout tabmorph kindex, kweightpoint, ktabnum1, ktabnum2, \  
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

## Initialization

*ifn1, ifn2 [, ifn3, ifn4,...ifnN]* - function table numbers. This is a set of chosen tables the user want to use in the morphing. All tables must have the same length. Be aware that only two of these tables can be chosen for the morphing at one time. Since it is possible to use non-integer numbers for the *ktabnum1* and *ktabnum2* arguments, the morphing is the result from the interpolation between adjacent consecutive tables of the set.

## Performance

*kout* - The output value for index *kindex*, resulting from morphing two tables (see below).

*kindex* - main index index of the morphed resultant table. The range is 0 to table\_length (not included).

*kweightpoint* - the weight of the influence of a pair of selected tables in the morphing. The range of this argument is 0 to 1. A zero makes it output the first table unaltered, a 1 makes it output the second table of the pair unaltered. All intermediate values between 0 and 1 determine the gradual morphing between the two tables of the pair.

*ktabnum1* - the first table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, the corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

*ktabnum2* - the second table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

The *tabmorph* family of opcodes is similar to the *table* family, but allows morphing between two tables chosen into a set of tables. Firstly the user has to provide a set of tables of equal length (*ifn2 [, ifn3, ifn4,...ifnN]*). Then he can choose a pair of tables in the set in order to perform the morphing: *ktabnum1* and *ktabnum2* are filled with numbers (zero represents the first table in the set, 1 the second, 2 the third and so on). Then determine the morphing between the two chosen tables with the *kweightpoint* parameter. After that the resulting table can be indexed with the *kindex* parameter like a normal table opcode. If the value of this parameter surpasses the length of tables (which must be the same for all tables), then it is wrapped around.

*tabmorph* acts similarly to the *table* opcode, that is, without using interpolation. This means that it trun-

cates the fractional part of the *kindex* argument. Anyway, fractional parts of *ktabnum1* and *ktabnum2* are significant, resulting in linear interpolation between the same element of two adjacent subsequent tables.

## See Also

*table*, *tabmorphi*, *tabmorpha*, *tabmorphak*, *ftmorf*,

## Credits

Author: Gabriel Maldonado

New in version 5.06

# tabmorpha

tabmorpha — Allow morphing between a set of tables at audio rate with interpolation.

## Description

*tabmorpha* allows morphing between a set of tables of the same size, by means of a weighted average between two currently selected tables.

## Syntax

```
aout tabmorpha aindex, aweightpoint, atabnum1, atabnum2, \  
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

## Initialization

*ifn1, ifn2, ifn3, ifn4, ... ifnN* - function table numbers. This is a set of chosen tables the user want to use in the morphing. All tables must have the same length. Be aware that only two of these tables can be chosen for the morphing at one time. Since it is possible to use non-integer numbers for the *atabnum1* and *atabnum2* arguments, the morphing is the result from the interpolation between adjacent consecutive tables of the set.

## Performance

*aout* - The output value for index *aindex*, resulting from morphing two tables (see below).

*aindex* - main index index of the morphed resultant table. The range is 0 to table\_length (not included).

*aweightpoint* - the weight of the influence of a pair of selected tables in the morphing. The range of this argument is 0 to 1. A zero makes it output the first table unaltered, a 1 makes it output the second table of the pair unaltered. All intermediate values between 0 and 1 determine the gradual morphing between the two tables of the pair.

*atabnum1* - the first table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, the corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

*atabnum2* - the second table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

The *tabmorpha* family of opcodes is similar to the *table* family, but allows morphing between two tables chosen into a set of tables. Firstly the user has to provide a set of tables of equal length (*ifn2* [, *ifn3*, *ifn4*,...*ifnN*]). Then he can choose a pair of tables in the set in order to perform the morphing: *atabnum1* and *atabnum2* are filled with numbers (zero represents the first table in the set, 1 the second, 2 the third and so on). Then determine the morphing between the two chosen tables with the *aweightpoint* parameter. After that the resulting table can be indexed with the *aindex* parameter like a normal table opcode. If the value of this parameter surpasses the length of tables (which must be the same for all tables), then it is wrapped around.

*tabmorpha* is the audio-rate version of *tabmorphi* (it uses interpolation). All input arguments work at a-

rate.

## See Also

*table, tabmorph, tabmorphi, tabmorphak, ftmorf,*

## Credits

Author: Gabriel Maldonado

New in version 5.06

# tabmorphak

tabmorphak — Allow morphing between a set of tables at audio rate with interpolation.

## Description

*tabmorphak* allows morphing between a set of tables of the same size, by means of a weighted average between two currently selected tables.

## Syntax

```
aout tabmorphak aindex, kweightpoint, ktabnum1, ktabnum2, \  
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

## Initialization

*ifn1, ifn2, ifn3, ifn4, ... ifnN* - function table numbers. This is a set of chosen tables the user want to use in the morphing. All tables must have the same length. Be aware that only two of these tables can be chosen for the morphing at one time. Since it is possible to use non-integer numbers for the *atabnum1* and *atabnum2* arguments, the morphing is the result from the interpolation between adjacent consecutive tables of the set.

## Performance

*aout* - The output value for index *aindex*, resulting from morphing two tables (see below).

*aindex* - main index index of the morphed resultant table. The range is 0 to *table\_length* (not included).

*kweightpoint* - the weight of the influence of a pair of selected tables in the morphing. The range of this argument is 0 to 1. A zero makes it output the first table unaltered, a 1 makes it output the second table of the pair unaltered. All intermediate values between 0 and 1 determine the gradual morphing between the two tables of the pair.

*ktabnum1* - the first table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, the corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

*ktabnum2* - the second table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

The *tabmorphak* family of opcodes is similar to the *table* family, but allows morphing between two tables chosen into a set of tables. Firstly the user has to provide a set of tables of equal length (*ifn2* [, *ifn3*, *ifn4*, ... *ifnN*]). Then he can choose a pair of tables in the set in order to perform the morphing: *atabnum1* and *atabnum2* are filled with numbers (zero represents the first table in the set, 1 the second, 2 the third and so on). Then determine the morphing between the two chosen tables with the *aweightpoint* parameter. After that the resulting table can be indexed with the *aindex* parameter like a normal table opcode. If the value of this parameter surpasses the length of tables (which must be the same for all tables), then it is wrapped around.

*tabmorphak* works at a-rate, but *kweightpoint*, *ktabnum1* and *ktabnum2* are working at k-rate, making it

more efficient than *tabmorpha*, since there are less calculations. Except the rate of these three arguments, it is identical to *tabmorpha*.

## See Also

*table*, *tabmorph*, *tabmorphi*, *tabmorpha*, *ftmorf*,

## Credits

Author: Gabriel Maldonado

New in version 5.06

# tabmorphi

tabmorphi — Allow morphing between a set of tables with interpolation.

## Description

*tabmorphi* allows morphing between a set of tables of the same size, by means of a weighted average between two currently selected tables.

## Syntax

```
kout tabmorphi kindex, kweightpoint, ktabnum1, ktabnum2, \  
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

## Initialization

*ifn1, ifn2 [, ifn3, ifn4,...ifnN]* - function table numbers. This is a set of chosen tables the user want to use in the morphing. All tables must have the same length. Be aware that only two of these tables can be chosen for the morphing at one time. Since it is possible to use non-integer numbers for the *ktabnum1* and *ktabnum2* arguments, the morphing is the result from the interpolation between adjacent consecutive tables of the set.

## Performance

*kout* - The output value for index *kindex*, resulting from morphing two tables (see below).

*kindex* - main index index of the morphed resultant table. The range is 0 to table\_length (not included).

*kweightpoint* - the weight of the influence of a pair of selected tables in the morphing. The range of this argument is 0 to 1. A zero makes it output the first table unaltered, a 1 makes it output the second table of the pair unaltered. All intermediate values between 0 and 1 determine the gradual morphing between the two tables of the pair.

*ktabnum1* - the first table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, the corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

*ktabnum2* - the second table chosen for the morphing. This number doesn't express the table number directly, but the position of the table in the set sequence (starting from 0 to N-1). If this number is an integer, corresponding table will be chosen unaltered. If it contains fractional values, then an interpolation with the next adjacent table will result.

The *tabmorphi* family of opcodes is similar to the *table* family, but allows morphing between two tables chosen into a set of tables. Firstly the user has to provide a set of tables of equal length (*ifn2 [, ifn3, ifn4,...ifnN]*). Then he can choose a pair of tables in the set in order to perform the morphing: *ktabnum1* and *ktabnum2* are filled with numbers (zero represents the first table in the set, 1 the second, 2 the third and so on). Then determine the morphing between the two chosen tables with the *kweightpoint* parameter. After that the resulting table can be indexed with the *kindex* parameter like a normal table opcode. If the value of this parameter surpasses the length of tables (which must be the same for all tables), then it is wrapped around.

*tabmorphi* is identical to *tabmorph*, but it performs linear interpolation for non-integer values of *kindex*,

much like *tablei*.

## See Also

*table*, *tabmorph*, *tabmorpha*, *tabmorphak*, *ftmorf*,

## Credits

Author: Gabriel Maldonado

New in version 5.06



# tabplay

tabplay — Playing-back control signals.

## Description

Plays-back control-rate signals on trigger-temporization basis.

## Syntax

```
tabplay ktrig, knumtics, kfn, kout1 [,kout2,..., koutN]
```

## Performance

*ktrig* -- starts playing when non-zero.

*knumtics* -- stop recording or reset playing pointer to zero when the number of tics defined by this argument is reached.

*kfn* -- table where k-rate signals are recorded.

*kout1,...,koutN* -- playback output signals.

The *tabplay* and *tabrec* opcodes allow to record/playback control signals on trigger-temporization basis.

*tabplay* plays back a group of k-rate signals, previously recorded by *tabrec* into a table. Each time *ktrig* argument is triggered, an internal counter is increased of one unit. After *knumtics* trigger impluses are received by *ktrig* argument, the internal counter is zeroed and playback is restarted from the beginning, in looping style.

These opcodes can be used like a sort of ``middle-term" memory that ``remembers" generated signals. Such memory can be used to supply generative music with a coherent iterative compositional structure.

## See Also

*tabrec*

## Credits

Written by Gabriel Maldonado.

# tabsum

tabsum — Addition des valeurs dans un intervalle d'une table.

## Description

Fait la somme des valeurs d'une f-table dans un intervalle contigu.

## Syntaxe

```
kr tabsum ifn[[, kmin] [, kmax]]
```

## Initialisation

*ifn* -- numéro de la table.

## Exécution

*kr* -- signal retourné.

*kmin*, *kmax* -- intervalle de la table à sommer. S'il est omis ou si les arguments sont nuls, il couvre par défaut les valeurs allant de 0 à la longueur de la table.

## Voir Aussi

Vectorial opcodes

## Crédits

Auteur : John ffitch  
Codemist Ltd  
2009

Nouveau dans la version 5.11

# tambourine

tambourine — Modèle semi-physique d'un son de tambourin.

## Description

*tambourine* est un modèle semi-physique d'un son de tambourin. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares tambourine kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \  
    [, ifreq1] [, ifreq2]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 32.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$\text{damping\_amount} = 0,9985 + (\text{idamp} * 0,002)$

La valeur par défaut de *damping\_amount* est 0,9985 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,75.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2300.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 5600.

*ifreq2* (facultatif) -- la deuxième fréquence de résonance. La valeur par défaut est 8100.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode *tambourine*. Il utilise le fichier *tambourine.csd* [examples/tambourine.csd].

## Exemple 520. Exemple de l'opcode tambourine.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tambourine.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1: An example of a tambourine.
instr 01
  al tambourine 15000, 0.01

  out al
endin

</CsInstruments>
<CsScore>

i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*bamboo, dripwater, guiro, sleighbells*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# tan

tan — Calcule une fonction tangente.

## Description

Retourne tangente de  $x$  ( $x$  en radians).

## Syntaxe

**tan**( $x$ ) (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode tan. Il utilise le fichier *tan.csd* [examples/tan.csd].

### Exemple 521. Exemple de l'opcode tan.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tan.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 25
  i1 = tan(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = -0.134
```

## Voir Aussi

*cos, cosh, cosinv, sin, sinh, sininv, tan, taninv*

## Crédits

Ecrit par John ffitch.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# tanh

tanh — Calcule une fonction tangente hyperbolique.

## Description

Retourne tangente hyperbolique de  $x$  ( $x$  en radians).

## Syntaxe

`tanh(x)` (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode tanh. Il utilise le fichier *tanh.csd* [examples/tanh.csd].

### Exemple 522. Exemple de l'opcode tanh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tanh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 1
  i1 = tanh(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsSoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.762
```

## Voir Aussi

*cos, cosh, cosinv, sin, sinh, sininv, tan, taninv*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.



# taninv

taninv — Calcule une fonction arctangente.

## Description

Retourne arctangente de  $x$  ( $x$  en radians).

## Syntaxe

**taninv**( $x$ ) (pas de restriction de taux)

## Exemples

Voici une exemple de l'opcode taninv. Il utilise le fichier *taninv.csd* [examples/taninv.csd].

### Exemple 523. Exemple de l'opcode taninv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o taninv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = taninv(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.464
```

## Voir Aussi

*cos, cosh, cosinv, sin, sinh, sininv, tan, tanh, taninv2*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

# taninv2

taninv2 — Retourne une tangente inverse (arctangente).

## Description

Retourne arctangente de  $iy/ix$ ,  $ky/kx$ , ou  $ay/ax$ .

## Syntax

ares **taninv2** ay, ax

ires **taninv2** iy, ix

kres **taninv2** ky, kx

Retourne arctangente de  $iy/ix$ ,  $ky/kx$ , ou  $ay/ax$ . Si  $y$  vaut zéro, *taninv2* retourne zéro quelque soit la valeur de  $x$ . Si  $x$  vaut zéro, la valeur de retour est :

- $\pi/2$ , si  $y$  est positif.
- $-\pi/2$ , si  $y$  est négatif.
- 0, si  $y$  vaut 0.

## Initialisation

*iy*, *ix* -- valeurs à transformer

## Exécution

*ky*, *kx* -- signaux de taux de contrôle à transformer

*ay*, *ax* -- signaux de taux audio à transformer

## Exemples

Voici un exemple de l'opcode taninv2. Il utilise le fichier *taninv2.csd* [examples/taninv2.csd].

### Exemple 524. Exemple de l'opcode taninv2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o taninv2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Returns the arctangent for 1/2.
i1 taninv2 1, 2

    print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.464
```

## Voir Aussi

*taninv*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.48 de Csound

Corrigé en mai 2002, grâce à Istvan Varga.

# tbvcf

tbvcf — Models some of the filter characteristics of a Roland TB303 voltage-controlled filter.

## Description

This opcode attempts to model some of the filter characteristics of a Roland TB303 voltage-controlled filter. Euler's method is used to approximate the system, rather than traditional filter methods. Cutoff frequency, Q, and distortion are all coupled. Empirical methods were used to try to unentwine, but frequency is only approximate as a result. Future fixes for some problems with this opcode may break existing orchestras relying on this version of *tbvcf*.

## Syntax

```
ares tbvcf asig, xfco, xres, kdist, kasym [, iskip]
```

## Initialization

*iskip* (optional, default=0) -- if non zero skip the initialisation of the filter. (New in Csound version 4.23f13 and 5.0)

## Performance

*asig* -- input signal. Should be normalized to  $\pm 1$ .

*xfco* -- filter cutoff frequency. Optimum range is 10,000 to 1500. Values below 1000 may cause problems.

*xres* -- resonance or Q. Typically in the range 0 to 2.

*kdist* -- amount of distortion. Typical value is 2. Changing *kdist* significantly from 2 may cause odd interaction with *xfco* and *xres*.

*kasym* -- asymmetry of resonance. Typically in the range 0 to 1.

## Examples

Here is an example of the *tbvcf* opcode. It uses the file *tbvcf.csd* [examples/tbvcf.csd].

### Exemple 525. Example of the *tbvcf* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o tbvcf.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

;-----
; TBVCF Test
; Coded by Hans Mikelson December, 2000
;-----

sr = 44100 ; Sample rate
ksmps = 10 ; Samples/Kontrol period
nchnls = 2 ; Normal stereo
0dbfs = 1
zakinit 50, 50

instr 10

idur = p3 ; Duration
iamp = p4 ; Amplitude
ifqc = cpspch(p5) ; Pitch to frequency
ipanl = sqrt(p6) ; Pan left
ipanr = sqrt(1-p6) ; Pan right
iq = p7
idist = p8
iasym = p9

kdcclk linseg 0, .002, 1, idur-.004, 1, .002, 0 ; Declick envelope

kfco expseg 10000, idur, 1000 ; Frequency envelope

ax vco 1, ifqc, 2, 0.5 ; Square wave

ay tbvcf ax, kfco, iq, idist, iasym ; TB-VCF
ay buthp ay/1, 100 ; Hi-pass

outs ay*iamp*ipanl*kdcclk, ay*iamp*ipanr*kdcclk
endin

</CsInstruments>
<CsScore>

f1 0 65536 10 1

; TeeBee Test
; Sta Dur Amp Pitch Pan Q Dist1 Asym
i10 0 0.2 0.5 7.00 .5 0.0 2.0 0.0
i10 0.3 0.2 0.5 7.00 .5 0.8 2.0 0.0
i10 0.6 0.2 0.5 7.00 .5 1.6 2.0 0.0
i10 0.9 0.2 0.5 7.00 .5 2.4 2.0 0.0
i10 1.2 0.2 0.5 7.00 .5 3.2 2.0 0.0
;i10 1.5 0.2 0.5 7.00 .5 4.0 2.0 0.0
i10 1.8 0.2 0.5 7.00 .5 0.0 2.0 0.25
i10 2.1 0.2 0.5 7.00 .5 0.8 2.0 0.25
i10 2.4 0.2 0.5 7.00 .5 1.6 2.0 0.25
i10 2.7 0.2 0.5 7.00 .5 2.4 2.0 0.25
i10 3.0 0.2 0.5 7.00 .5 3.2 2.0 0.25
i10 3.3 0.2 0.5 7.00 .5 4.0 2.0 0.25
i10 3.6 0.2 0.5 7.00 .5 0.0 2.0 0.5
i10 3.9 0.2 0.5 7.00 .5 0.8 2.0 0.5
i10 4.2 0.2 0.5 7.00 .5 1.6 2.0 0.5
i10 4.5 0.2 0.5 7.00 .5 2.4 2.0 0.5
i10 4.8 0.2 0.5 7.00 .5 3.2 2.0 0.5
i10 5.1 0.2 0.5 7.00 .5 4.0 2.0 0.5
i10 5.4 0.2 0.5 7.00 .5 0.0 2.0 0.55
i10 5.7 0.2 0.5 7.00 .5 0.8 2.0 0.75
i10 6.0 0.2 0.5 7.00 .5 1.6 2.0 0.75
i10 6.3 0.2 0.5 7.00 .5 2.4 2.0 0.75
i10 6.6 0.2 0.5 7.00 .5 3.2 2.0 0.75
i10 6.9 0.2 0.5 7.00 .5 4.0 2.0 0.75
i10 7.2 0.2 0.5 7.00 .5 0.0 2.0 1.0
i10 7.5 0.2 0.5 7.00 .5 0.8 2.0 1.0
i10 7.8 0.2 0.5 7.00 .5 1.6 2.0 1.0
i10 8.1 0.2 0.5 7.00 .5 2.4 2.0 1.0
i10 8.4 0.2 0.5 7.00 .5 3.2 2.0 1.0
i10 8.7 0.2 0.5 7.00 .5 4.0 2.0 1.0
e

</CsScore>
</CsSoundSynthesizer>

```

## Credits

Author: Hans Mikelson  
December, 2000 -- January, 2001

New in Csound 4.10

# tempest

tempest — Estimate the tempo of beat patterns in a control signal.

## Description

Estimate the tempo of beat patterns in a control signal.

## Syntax

```
ktemp tempest kin, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, \  
istartempo, ifn [, idisprd] [, itweek]
```

## Initialization

*iprd* -- period between analyses (in seconds). Typically about .02 seconds.

*imindur* -- minimum duration (in seconds) to serve as a unit of tempo. Typically about .2 seconds.

*imemdur* -- duration (in seconds) of the *kin* short-term memory buffer which will be scanned for periodic patterns. Typically about 3 seconds.

*ihp* -- half-power point (in Hz) of a low-pass filter used to smooth input *kin* prior to other processing. This will tend to suppress activity that moves much faster. Typically 2 Hz.

*ithresh* -- loudness threshold by which the low-passed *kin* is center-clipped before being placed in the short-term buffer as tempo-relevant data. Typically at the noise floor of the incoming data.

*ihtim* -- half-time (in seconds) of an internal forward-masking filter that masks new *kin* data in the presence of recent, louder data. Typically about .005 seconds.

*ixfdbak* -- proportion of this unit's *anticipated value* to be mixed with the incoming *kin* prior to all processing. Typically about .3.

*istartempo* -- initial tempo (in beats per minute). Typically 60.

*ifn* -- table number of a stored function (drawn left-to-right) by which the short-term memory data is attenuated over time.

*idisprd* (optional) -- if non-zero, display the short-term past and future buffers every *idisprd* seconds (normally a multiple of *iprd*). The default value is 0 (no display).

*itweek* (optional) -- fine-tune adjust this unit so that it is stable when analyzing events controlled by its own output. The default value is 1 (no change).

## Performance

*tempest* examines *kin* for amplitude periodicity, and estimates a current tempo. The input is first low-pass filtered, then center-clipped, and the residue placed in a short-term memory buffer (attenuated over time) where it is analyzed for periodicity using a form of autocorrelation. The period, expressed as a *tempo* in beats per minute, is output as *ktemp*. The period is also used internally to make predictions about future amplitude patterns, and these are placed in a buffer adjacent to that of the input. The two adjacent buffers can be periodically displayed, and the predicted values optionally mixed with the inco-



ming signal to simulate expectation.

This unit is useful for sensing the metric implications of any k-signal (e.g.- the RMS of an audio signal, or the second derivative of a conducting gesture), before sending to a *tempo* statement.

## Examples

Here is an example of the *tempest* opcode. It uses the file *tempest.csd* [examples/tempest.csd], and *beats.wav* [examples/beats.wav].

### Exemple 526. Example of the *tempest* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o tempest.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use the "beats.wav" sound file.
asig soundin "beats.wav"
; Extract the pitch and the envelope.
kcps, krms pitchamdf asig, 150, 500, 200

iprd = 0.01
imindur = 0.1
imemdur = 3
ihp = 1
ithresh = 30
ihtim = 0.005
ixfdbak = 0.05
istartempo = 110
ifn = 1

; Estimate its tempo.
k1 tempest krms, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, istartempo, ifn
printk2 k1

out asig
endin

</CsInstruments>
<CsScore>

; Table #1, a declining line.
f 1 0 128 16 1 128 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

The tempo of the audio file « beats.wav » is 120 beats per minute. In this examples, tempest will print out its best guess as the audio file plays. Its output should include lines like this:

```
. i1 118.24654  
. i1 121.72949
```

# tempo

tempo — Apply tempo control to an uninterpreted score.

## Description

Apply tempo control to an uninterpreted score.

## Syntax

```
tempo ktempo, istartempo
```

## Initialization

*istartempo* -- initial tempo (in beats per minute). Typically 60.

## Performance

*ktempo* -- The tempo to which the score will be adjusted.

*tempo* allows the performance speed of Csound scored events to be controlled from within an orchestra. It operates only in the presence of the Csound *-t* flag. When that flag is set, scored events will be performed from their uninterpreted p2 and p3 (beat) parameters, initially at the given command-line tempo. When a *tempo* statement is activated in any instrument (*ktempo* 0.), the operating tempo will be adjusted to *ktempo* beats per minute. There may be any number of *tempo* statements in an orchestra, but coincident activation is best avoided.

## Examples

Here is an example of the tempo opcode. Remember, it only works if you use the *-t* flag with Csound. The example uses the file *tempo.csd* [examples/tempo.csd].

### Exemple 527. Example of the tempo opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tempo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
```

```
kval tempoval

printk 0.1, kval

; If the fourth p-field is 1, increase the tempo.
if (p4 == 1) kgoto speedup
    kgoto playit

speedup:
    ; Increase the tempo to 150 beats per minute.
    tempo 150, 60

playit:

    a1 oscil 10000, 440, 1
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = plays at a faster tempo (when p4=1).
; Play Instrument #1 at the normal tempo, repeat 3 times.
r3
i 1 00.00 00.25 0
i 1 00.25 00.25 0
i 1 00.50 00.25 0
i 1 00.75 00.25 0
s

; Play Instrument #1 at a faster tempo, repeat 3 times.
r3
i 1 00.00 00.25 1
i 1 00.25 00.25 0
i 1 00.50 00.25 0
i 1 00.75 00.25 0
s

e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*tempoval*

## Credits

Example written by Kevin Conder.

# tempoval

tempoval — Reads the current value of the tempo.

## Description

Reads the current value of the tempo.

## Syntax

```
kres tempoval
```

## Performance

*kres* -- the value of the tempo. If you use a positive value with the *-t* command-line flag, *tempoval* returns the percentage increase/decrease from the original tempo of 60 beats per minute. If you don't, its value will be 60 (for 60 beats per minute).

## Examples

Here is an example of the *tempoval* opcode. Remember, it only works if you use the *-t* flag with Csound. It uses the file *tempoval.csd* [examples/tempoval.csd].

### Exemple 528. Example of the tempoval opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o tempoval.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Adjust the tempo to 120 beats per minute.
tempo 120, 60

; Get the tempo value.
kval tempoval

printks "kval = %f\\n", 0.1, kval
endin

</CsInstruments>
<CsScore>
```

```
; Play Instrument #1 for one second.  
i 1 0 1  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Since 120 beats per minute is a 50% increase over the original 60 beats per minute, its output should include lines like:

```
kval = 0.500000
```

## See Also

*tempo* and *miditempo*

## Credits

Example written by Kevin Conder.

New in version 4.15

December 2002. Thanks to Drake Wilson for pointing out unclear documentation.

# tigoto

tigoto — Transfère le contrôle lors de la phase d'initialisation si la nouvelle note est liée à la précédente note tenue.

## Description

Semblable à *igoto* mais ne fonctionne que lors d'une phase d'initialisation concernant une nouvelle note « liée » à une note précédente tenue. (Voir l'*instruction i*). Ca ne fonctionne pas s'il n'y a pas de liaison. Permet à un instrument d'ignorer l'initialisation de ses unités si une liaison a été proposée avec succès. (Voir aussi *tival*).

## Syntaxe

`tigoto label`

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.

## Voir Aussi

*cigoto, goto, if, igoto, kgoto, timeout*

# timedseq

timedseq — Time Variant Sequencer

## Description

An event-sequencer in which time can be controlled by a time-pointer. Sequence data are stored into a table.

## Syntax

```
ktrig timedseq ktimpnt, ifn, kp1 [,kp2, kp3, ...,kpN]
```

## Initialization

*ifn* -- number of table containing sequence data.

## Performance

*ktri* -- output trigger signal

*ktimpnt* -- time pointer into sequence file, in seconds.

*kp1,...,kpN* -- output p-fields of notes. *kp2* meaning is relative action time and *kp3* is the duration of notes in seconds.

*timedseq* is a sequencer that allows to schedule notes starting from a user sequence, and depending from an external timing given by a time-pointer value (*ktimpnt* argument). User should fill table *ifn* with a list of notes, that can be provided in an external text file by using GEN23, or by typing it directly in the orchestra (or score) file with GEN02. The format of the text file containing the sequence is made up simply by rows containing several numbers separated by space (similarly to normal Csound score). The first value of each row must be a positive or null value, except for a special case that will be explained below. This first value is normally used to define the instrument number corresponding to that particular note (like normal score). The second value of each row must contain the action time of corresponding note and the third value its duration. This is an example:

```
0 0      0.25 1  93
0 0.25  0.25 2  63
0 0.5   0.25 3  91
0 0.75  0.25 4  70
0 1     0.25 5  83
0 1.25  0.25 6  75
0 1.5   0.25 7  78
0 1.75  0.25 8  78
0 2     0.25 9  83
0 2.25  0.25 10 70
0 2.5   0.25 11 54
0 2.75  0.25 12 80
-1 3    -1   -1 -1 ;; last row of the sequence
```

In this example, the first value of each row is always zero (it is a dummy value, but this p-field can be used, for example, to express a MIDI channel or an instrument number), except the last row, that begins with -1. This value (-1) is a special value, that indicates the end of sequence. It has itself an action time, because sequences can be looped. So the previous sequence has a default duration of 3 seconds, being value 3 the last action time of the sequence.



It is important that ALL lines contains the same number of values (in the example all rows contains exactly 5 values). The number of values contained by each row, MUST be the number of kpXX output arguments (notice that, even if kp1, kp2 etc. are placed at the right of the opcode, they are output arguments, not input arguments).

ktimpnt argument provide the real temporization of the sequence. Actually the passage of time through sequence is specified by ktimpnt itself, which represents the time in seconds. ktimpnt must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the sequence file, in the same way of pvoc or lpread. When ktimpnt crosses the action time of a note, a trigger signal is sent to ktrig output argument, and kp1, kp2,...kpN arguments are updated with the values of that note. This information can then be used with schedk or schedkwhen to actually activate note events. Notice that kp1,...kpn data can be further processed (for example delayed with delayk, transposed, etc.) before feeding schedk or schedkwhen.

timepoint can be controlled by linear signal, for example:

```
ktimpnt line      0,p3,3 ; original sequence duration was 3 secs
ktrig   timedseq ktimpnt,1,kp1,kp2,kp3,kp4,kp5
        schedk   ktrig, 105, 2, 0, kp3,kp4,kp5
```

in this case the complete sequence (with original duration of 3 seconds) will be played in p3 seconds.

You can loop a sequence by controlling it with a phasor:

```
kphs     phasor    1/3
ktimpnt  =         kphs * 3
ktrig    timedseq ktimpnt,1,kp1,kp2,kp3,kp4,kp5
        schedk   ktrig, 105, 2, 0, kp3,kp4,kp5
```

Obviously you can play only a fragment of the sequence, read it backward, and non-linearly access sequence data in the same way of pvoc and lpread opcodes.

With timedseq opcode you can do almost all things of a normal score, except you have the following limitations: 1. You can't have two notes exactly starting with the same action time; actually at least a k-cycle should separate timing of two notes (otherwise the schedk mechanism eats one of them). 2. all notes of the sequence must have the same number of p-fields (even if they activate different instruments). You can remedy this limitation by filling with dummy values notes that belongs to instruments with less p-fields than other ones.

## See Also

*GEN02, GEN23, seqtime, seqtime2, trigseq*

## Credits

Author: Gabriel Maldonado

# timeinstk

timeinstk — Lit le temps absolu en cycles de taux-k.

## Description

Lit le temps absolu en cycles de taux-k, depuis le démarrage d'une instance d'un instrument. Appelé aussi bien lors de la phase d'initialisation que pendant la phase d'exécution.

## Syntaxe

```
kres timeinstk
```

## Exécution

*timeinstk* donne le temps en cycles de taux-k. Ainsi avec :

```
sr    = 44100
kr    = 6300
ksmps = 7
```

après une demi-seconde, l'opcode *timeinstk* retournera 3150. Il retourne toujours un nombre entier.

*timeinstk* produit une variable de taux-k en sortie. Il n'y a pas de paramètres d'entrée.

*timeinstk* est semblable à *timek* sauf qu'il retourne le temps écoulé depuis le démarrage de cette instance de l'instrument.

## Exemples

Voici un exemple de l'opcode *timeinstk*. Il utilise le fichier *timeinstk.csd* [examples/timeinstk.csd].

### Exemple 529. Exemple de l'opcode timeinstk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o timeinstk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
; Print out the value from timeinstk every half-second.
k1 timeinstk
printks "k1 = %f samples\\n", 0.5, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
k1 = 1.000000 samples
k1 = 2205.000000 samples
k1 = 4410.000000 samples
k1 = 6615.000000 samples
k1 = 8820.000000 samples
```

## Voir Aussi

*timeinsts, timek, times*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Exemple écrit par Kevin Conder.

# timeinsts

timeinsts — Lit le temps absolu en secondes.

## Description

Lit le temps absolu en secondes, depuis le démarrage d'une instance d'un instrument.

## Syntaxe

```
kres timeinsts
```

## Exécution

Le temps en secondes est donné par *timeinsts*. Il retournera 0.5 après une demi-seconde.

*timeinsts* produit une variable de taux-k en sortie. Il n'y a pas de paramètres d'entrée.

*timeinsts* est semblable à *times* sauf qu'il retourne le temps écoulé depuis le démarrage de cette instance de l'instrument.

## Exemples

Voici un exemple de l'opcode *timeinsts*. Il utilise le fichier *timeinsts.csd* [examples/timeinsts.csd].

### Exemple 530. Exemple de l'opcode *timeinsts*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o timeinsts.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the value from timeinsts every half-second.
k1 timeinsts
printks "k1 = %f seconds\\n", 0.5, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
```

```
i 1 0 2  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1 = 0.000227 seconds  
k1 = 0.500000 seconds  
k1 = 1.000000 seconds  
k1 = 1.500000 seconds  
k1 = 2.000000 seconds
```

## Voir Aussi

*timeinstk, timek, times*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Exemple écrit par Kevin Conder.

# timek

timek — Lit le temps absolu en cycles de taux-k.

## Description

Lit le temps absolu en cycles de taux-k, depuis le début de l'exécution.

## Syntaxe

```
ires timek
```

```
kres timek
```

## Exécution

*timek* donne le temps en cycles de taux-k. Ainsi avec :

```
sr      = 44100  
kr      = 6300  
ksmps = 7
```

après une demi-seconde, l'opcode *timek* retournera 3150. Il retourne toujours un nombre entier.

*timek* produit une variable de taux-k en sortie. Il n'y a pas de paramètres d'entrée.

*timek* peut aussi opérer seulement au démarrage de l'instance de l'instrument. Il produit alors une variable de taux-i (préfixée par *i* ou *gi*) en sortie.

## Exemples

Voici un exemple de l'opcode *timek*. Il utilise le fichier *timek.csd* [examples/timek.csd].

### Exemple 531. Exemple de l'opcode *timek*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  
-odac          -iadc      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o timek.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
; Initialize the global variables.  
sr = 44100  
kr = 4410  
ksmps = 10  
nchnls = 1
```

```
; Instrument #1.
instr 1
; Print out the value from timek every half-second.
k1 timek
printks "k1 = %f samples\\n", 0.5, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1 = 1.000000 samples
k1 = 2205.000000 samples
k1 = 4410.000000 samples
k1 = 6615.000000 samples
k1 = 8820.000000 samples
```

## Voir Aussi

*timeinstk, timensts, times*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# times

times — Lit le temps absolu en secondes.

## Description

Lit le temps absolu en secondes, depuis le début de l'exécution.

## Syntaxe

```
ires times
```

```
kres times
```

## Exécution

Le temps en secondes est donné par *times*. Il retournera 0.5 après une demi-seconde.

*times* produit une variable de taux-k en sortie. Il n'y a pas de paramètres d'entrée.

*times* peut aussi opérer au démarrage de l'instance de l'instrument. Il produit alors une variable de taux-i (préfixée par *i* ou *gi*) en sortie.

## Exemples

Voici un exemple de l'opcode *times*. Il utilise le fichier *times.csd* [examples/times.csd].

### Exemple 532. Exemple de l'opcode *times*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o times.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the value from times every half-second.
k1 times
printks "k1 = %f seconds\\n", 0.5, k1
endin

</CsInstruments>
```



```
<CsScore>
; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1 = 0.000227 seconds
k1 = 0.500000 seconds
k1 = 1.000000 seconds
k1 = 1.500000 seconds
k1 = 2.000000 seconds
```

## Voir Aussi

*timeinstk, timeinsts, timek*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Exemple écrit par Kevin Conder.

# timeout

timeout — Branchement conditionnel durant l'exécution en fonction de la durée de la note qui s'est déjà écoulée.

## Description

Branchement conditionnel durant l'exécution en fonction de la durée de la note qui s'est déjà écoulée. *istrt* et *idur* sont exprimés en secondes. Le branchement vers *label* aura lieu à partir de l'instant *istrt*, et restera actif pendant *idur* secondes. Noter que *timeout* peut être réinitialisé pour des activations multiples dans une seule note (voir l'exemple de *reinit*).

## Syntaxe

```
timeout istrt, idur, label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.

## Voir Aussi

*goto*, *if*, *igoto*, *kgoto*, *tigoto*

# tival

tival — Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i.

## Syntaxe

```
ir tival
```

## Description

Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i.

## Initialisation

Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i. Affecte 1 si la note est « liée » à une note tenue précédente (voir l'*instruction i*) ; affecte 0 s'il n'y a pas de liaison. (Voir aussi *tigoto*.)

## Voir Aussi

`=`, `divz`, `init`

# tlineto

tlineto — Generate glissandos starting from a control signal.

## Description

Generate glissandos starting from a control signal with a trigger.

## Syntax

```
kres tlineto ksig, ktime, ktrig
```

## Performance

*kres* -- Output signal.

*ksig* -- Input signal.

*ktime* -- Time length of glissando in seconds.

*ktrig* -- Trigger signal.

*tlineto* is similar to *lineto* but can be applied to any kind of signal (not only stepped signals) without producing discontinuities. Last value of each segment is sampled and held from input signal each time *ktrig* value is set to a nonzero value. Normally *ktrig* signal consists of a sequence of zeroes (see *trigger opcode*).

The effect of glissando is quite different from *port*. Since in these cases, the lines are straight. Also the context of useage is different.

## See Also

*lineto*

## Credits

Author: Gabriel Maldonado

New in Version 4.13

# tone

tone — Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

## Description

Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

*tone* est filtre RII à un terme. Sa formule est :

$$y_n = c1 * x_n + c2 * y_{n-1}$$

où

- $b = 2 - \cos(2 \# \text{hp/sr})$ ;
- $c2 = b - \sqrt{b^2 - 1.0}$
- $c1 = 1 - c2$

## Syntaxe

```
ares tone asig, khp [, iskip]
```

## Initialisation

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ares* -- le signal audio de sortie.

*asig* -- le signal audio en entrée.

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

*tone* implémente un filtre passe-bas récursif du premier ordre dans lequel la variable *khp* (en Hz) détermine le point à mi-puissance de la courbe de réponse. La mi-puissance est définie par puissance maximale / racine de 2.

## Voir Aussi

*areson*, *aresonk*, *atone*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tonek*

# tonek

tonek — Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

## Description

Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

## Syntaxe

```
kres tonek ksig, khp [, iskip]
```

## Initialisation

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

*tonek* est semblable à *tone* à part le fait que sa sortie se fait au taux de contrôle plutôt qu'au taux audio.

## Voir Aussi

*areson*, *aresonk*, *atone*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tone*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# tonex

tonex — Emule une série de filtres utilisant l'opcode *tone*.

## Description

*tonex* est équivalent à un filtre constitué de plusieurs couches de filtres *tone* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupe plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de *k* cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

## Syntaxe

```
ares tonex asig, khp [, inumlayer] [, iskip]
```

## Initialisation

*inumlayer* (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

## Voir Aussi

*atonex*, *resonx*

## Crédits

Auteur : Gabriel Maldonado (adapté par John ffitich)  
Italie

Nouveau dans la version 3.49 de Csound

# trandom

**trandom** — Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale en fonction d'un déclencheur.

## Description

Génère au taux-*k* une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale chaque fois que le paramètre de déclenchement est différent de 0.

## Syntaxe

```
kout trandom ktrig, kmin, kmax
```

## Exécution

*ktrig* -- déclencheur (l'opcode produit un nouveau nombre aléatoire chaque fois que cette valeur est différente de 0.

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

*trandom* est presque identique à l'opcode *random* sauf que *trandom* ne renouvelle sa sortie avec une nouvelle valeur aléatoire que si l'argument *ktrig* est déclenché (c-à-d chaque fois qu'il est différent de zéro).

## Voir Aussi

*random*

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5.06



# tradsyn

tradsyn — Streaming partial track additive synthesis

## Description

The `tradsyn` opcode takes an input containing a TRACKS pv streaming signal (as generated, for instance by `partials`), as described in Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc. of ICMC05, Barcelona. It resynthesises the signal using linear amplitude and frequency interpolation to drive a bank of interpolating oscillators with amplitude and pitch scaling controls.

## Syntax

```
asig tradsyn fin, kscal, kpitch, kmaxtracks, ifn
```

## Performance

*asig* -- output audio rate signal

*fin* -- input pv stream in TRACKS format

*kscal* -- amplitude scaling

*kpitch* -- pitch scaling

*kmaxtracks* -- max number of tracks in resynthesis. Limiting this will cause a non-linear filtering effect, by discarding newer and higher-frequency tracks (tracks are ordered by start time and ascending frequency, respectively)

*ifn* -- function table containing one cycle of a sinusoid (sine or cosine)

## Examples

### Exemple 533. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
aout tradsyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
out aout
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis with pitch shifting.

## Credits

Author: Victor Lazzarini  
June 2005

New plugin in version 5

November 2004.

# transeg

transeg — Construit une enveloppe définie par l'utilisateur.

## Description

Construit une enveloppe définie par l'utilisateur.

## Syntaxe

```
ares transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

```
kres transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

## Initialisation

*ia* -- valeur de départ.

*ib*, *ic*, etc. -- valeur après *idur* secondes.

*idur* -- durée en secondes du premier segment. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

*idur2*,...*idurx* etc. -- durée en secondes de chaque segment.

*itype*, *itype2*, etc. -- s'il vaut 0, un segment de droite est produit. S'il est différent de 0, *transeg* crée la courbe suivante en *n* pas :

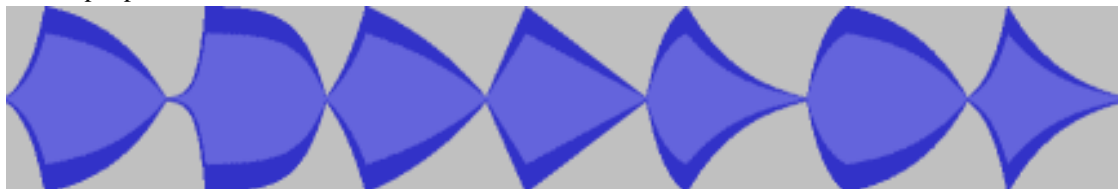
$$\text{ibeg} + (\text{ivalue} - \text{ibeg}) * (1 - \exp(i * \text{itype} / (n - 1))) / (1 - \exp(\text{itype}))$$

## Exécution

Si *type* > 0, on a une courbe montant lentement (concave) ou décroissant lentement (convexe), tandis que si *type* < 0, la courbe monte rapidement (convexe) ou décroît rapidement (concave). Voir aussi *GEN16*.

## Exemples

Voici un exemple de l'opcode transeg. Il utilise le fichier *transeg.csd* [examples/transeg.csd]. L'exemple produit la sortie suivante :



Sortie de l'exemple de transeg.

### Exemple 534. Exemple de l'opcode transeg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o transeg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

0dbfs = 1

instr 1
; p4 and p5 determine the type of curve for each
; section of the envelope
kenv transeg 0.01, p3*0.25, p4, 1, p3*0.75, p5, 0.01
a1 oscil kenv, 440, 1
outs a1, a1
endin

</CsInstruments>
<CsScore>
; Table #1, a sine wave.
f 1 0 16384 10 1

i 1 0 2 2 2
i 1 + . 5 5
i 1 + . 1 1
i 1 + . 0 0
i 1 + . -2 -2
i 1 + . -2 2
i 1 + . 2 -2
e
</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*expsega, expsegr, linseg, linsegr, transegr.*

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Octobre 2000

Nouveau dans la version 4.09 de Csound

Merci à Matt Gerassimoff pour avoir précisé la syntaxe correcte de la commande.

# transegr

transegr — Construit une enveloppe définissable par l'utilisateur prolongée par un segment de relâchement.

## Description

Construit une enveloppe définissable par l'utilisateur. Semblable à *transeg*, avec un segment de relâchement en prolongement.

## Syntaxe

```
ares transegr ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

```
kres transegr ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

## Initialisation

*ia* -- valeur de départ.

*ib*, *ic*, etc. -- valeur après *idur* secondes.

*idur* -- durée en secondes du premier segment. Avec une valeur nulle ou négative toute initialisation sera ignorée.

*idur2*,... *idurx* etc. -- durée de segment en secondes.

*itype*, *itype2*, etc. -- s'il vaut 0, un segment de droite est produit. S'il est non nul, alors *transegr* crée la courbe suivante pour *n* pas :

$$\text{ibeg} + (\text{ivalue} - \text{ibeg}) * (1 - \exp(i * \text{itype} / (n - 1))) / (1 - \exp(\text{itype}))$$

## Exécution

Si *itype* > 0, il y a une courbe croissant lentement (concave) ou décroissant lentement (convexe), tandis que si *itype* < 0, la courbe est à croissance rapide (convexe) ou à décroissance rapide (concave). Voir aussi *GEN16*.

Cet opcode est le même que *transeg* avec un segment de relâchement additionnel déclenché par un événement MIDI noteoff, un *événement de note* avec p1 négatif dans la partition ou un opcode *turnoff2*.

## Voir Aussi

*expsega*, *expsegr*, *linseg*, *linsegr*, *transeg*

## Crédits

Auteur : John ffitich

Janvier 2010

Nouveau dans la version 5.12 de Csound.

# trcross

trcross — Streaming partial track cross-synthesis.

## Description

The trcross opcode takes two inputs containing TRACKS pv streaming signals (as generated, for instance by partials) and cross-synthesises them into a single TRACKS stream. Two different modes of operation are used: mode 0, cross-synthesis by multiplication of the amplitudes of the two inputs and mode 1, cross-synthesis by the substitution of the amplitudes of input 1 by the input 2. Frequencies and phases of input 1 are preserved in the output. The cross-synthesis is done by matching tracks between the two inputs using a 'search interval'. The matching algorithm will look for tracks in the second input that are within the search interval around each track in the first input. This interval can be changed at the control rate. Wider search intervals will find more matches.

## Syntax

```
fsig trcross fin1, fin2, ksearch,kdepth[,kmode]
```

## Performance

*fsig* -- output pv stream in TRACKS format

*fin1* -- first input pv stream in TRACKS format.

*fin2* -- second input pv stream in TRACKS format

*ksearch* -- search interval ratio, defining a 'search area' around each track of 1st input for matching purposes.

*kdepth* -- depth of effect (0-1).

*kmode* -- mode of cross-synthesis. 0, multiplication of amplitudes (filtering), 1, substitution of amplitudes of input 1 by input 2 (akin to vocoding). Defaults to 0.

## Examples

### Exemple 535. Example

```
ain inch 1 ; input signals
ain inch 2
fstl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fstl,fsi2,.003,1,3,500 ; partial tracking
fstl1,fsi12 pvsifd ain,2048,512,1 ; ifd analysis (second input)
fstl1 partials fstl1,fsi12,.003,1,3,500 ; partial tracking \ (second input)
fcr trcross fst,fstl1, 1.05, 1 ; cross-synthesis (mode 0)
aout tradsyn fcr, 1, 1, 500, 1 ; resynthesis of tracks
out aout
```

The example above shows partial tracking of two ifd-analysis signals, cross-synthesis, followed by the

remix of the two parts of the spectrum and resynthesis.

## Credits

Author: Victor Lazzarini  
February 2006

New in Csound5.01



# trfilter

trfilter — Streaming partial track filtering.

## Description

The trfilter opcode takes an input containing a TRACKS pv streaming signal (as generated, for instance by partials) and filters it using an amplitude response curve stored in a function table. The function table can have any size (no restriction to powers-of-two). The table lookup is done by linear-interpolation. It is possible to create time-varying filter curves by updating the amplitude response table with a table-writing opcode.

## Syntax

```
fsig trfilter fin, kamnt, ifn
```

## Performance

*fsig* -- output pv stream in TRACKS format

*fin* -- input pv stream in TRACKS format

*kamnt* -- amount of filtering (0-1)

*ifn* -- function table number. This will contain an amplitude response curve, from 0 Hz to the Nyquist (table indexes 0 to N). Any size is allowed. Larger tables will provide a smoother amplitude response curve. Table reading uses linear interpolation.

## Examples

### Exemple 536. Example

```
gifn ftgen 2, 0, -22050, 5 1 1000 1 4000 0.000001 17050 0.000001 ; low-pass filter curve of 22050 points
instr 1
  ain inch 1 ; input signal
  fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
  fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
  fscl trfilter fst, 1, gifn ; filtering using function table 2
  aout tradesyn fscl, 1, 1, 500, 1 ; resynthesis
out aout
endin
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis with low-pass filtering.

## Credits

Author: Victor Lazzarini;  
February 2006

New in Csound5.01

# trhighest

trhighest — Extracts the highest-frequency track from a streaming track input signal.

## Description

The trhighest opcode takes an input containing TRACKS pv streaming signals (as generated, for instance by partials) and outputs only the highest track. In addition it outputs two k-rate signals, corresponding to the frequency and amplitude of the highest track signal.

## Syntax

```
fsig, kfr, kamp trhighest finl, kscal
```

## Performance

*fsig* -- output pv stream in TRACKS format

*kfr* -- frequency (in Hz) of the highest-frequency track

*kamp* -- amplitude of the highest-frequency track

*fin* -- input pv stream in TRACKS format.

*kscal* -- amplitude scaling of output.

## Examples

### Exemple 537. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
fhi,kfr,kamp trhighest fst,1 ; highest freq-track
aout tradsyn fhi, 1, 1, 1, 1 ; resynthesis of highest frequency
out aout
```

The example above shows partial tracking of an ifd-analysis signal, extraction of the highest frequency and resynthesis.

## Credits

Author: Victor Lazzarini  
February 2006

New in Csound5.01

# trigger

trigger — Informs when a krate signal crosses a threshold.

## Description

Informs when a krate signal crosses a threshold.

## Syntax

kout **trigger** ksig, kthreshold, kmode

## Performance

*ksig* -- input signal

*kthreshold* -- trigger threshold

*kmode* -- can be 0 , 1 or 2

Normally *trigger* outputs zeroes: only each time *ksig* crosses *kthreshold* *trigger* outputs a 1. There are three modes of using *ktrig*:

- *kmode* = 0 - (down-up) *ktrig* outputs a 1 when current value of *ksig* is higher than *kthreshold*, while old value of *ksig* was equal to or lower than *kthreshold*.
- *kmode* = 1 - (up-down) *ktrig* outputs a 1 when current value of *ksig* is lower than *kthreshold* while old value of *ksig* was equal or higher than *kthreshold*.
- *kmode* = 2 - (both) *ktrig* outputs a 1 in both the two previous cases.

## Examples

Here is an example of the trigger opcode. It uses the file *trigger.csd* [examples/trigger.csd].

### Exemple 538. Example of the trigger opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o trigger.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```

kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a square-wave low frequency oscillator as the trigger.
klf lfo 1, 10, 3
ktr trigger klf, 1, 2

; When the value of the trigger isn't equal to 0, print it out.
if (ktr == 0) kgoto contin
; Print the value of the trigger and the time it occurred.
ktm times
printks "time = %f seconds, trigger = %f\\n", 0, ktm, ktr

contin:
; Continue with processing.
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Its output should include lines like this:

```

time = 0.050340 seconds, trigger = 1.000000
time = 0.150340 seconds, trigger = 1.000000
time = 0.250340 seconds, trigger = 1.000000
time = 0.350340 seconds, trigger = 1.000000
time = 0.450340 seconds, trigger = 1.000000
time = 0.550340 seconds, trigger = 1.000000
time = 0.650340 seconds, trigger = 1.000000
time = 0.750340 seconds, trigger = 1.000000
time = 0.850340 seconds, trigger = 1.000000
time = 0.950340 seconds, trigger = 1.000000

```

## Credits

Author: Gabriel Maldonado  
Italy

Example written by Kevin Conder.

New in Csound version 3.49

# trigseq

trigseq — Accepts a trigger signal as input and outputs a group of values.

## Description

Accepts a trigger signal as input and outputs a group of values.

## Syntax

```
trigseq ktrig_in, kstart, kloop, kinitndx, kfn_values, kout1 [, kout2] [...]
```

## Performance

*ktrig\_in* -- input trigger signal

*kstart* -- start index of looped section

*kloop* -- end index of looped section

*kinitndx* -- initial index



### Note

Although *kinitndx* is listed as k-rate, it is in fact accessed only at init-time. So if you are using a k-rate argument, it must be assigned with *init*.

*kfn\_values* -- numer of a table containing a sequence of groups of values

*kout1* -- output values

*kout2*, ... (optional) -- more output values

This opcode handles timed-sequences of groups of values stored into a table.

*trigseq* accepts a trigger signal (*ktrig\_in*) as input and outputs group of values (contained in the *kfn\_values* table) each time *ktrig\_in* assumes a non-zero value. Each time a group of values is triggered, table pointer is advanced of a number of positions corresponding to the number of group-elements, in order to point to the next group of values. The number of elements of groups is determined by the number of *koutX* arguments.

It is possible to start the sequence from a value different than the first, by assigning to *initndx* an index different than zero (which corresponds to the first value of the table). Normally the sequence is looped, and the start and end of loop can be adjusted by modifying *kstart* and *kloop* arguments. User must be sure that values of these arguments (as well as *kinitndx*) correspond to valid table numbers, otherwise Csound will crash because no range-checking is implemented.

It is possible to disable loop (one-shot mode) by assigning the same value both to *kstart* and *kloop* arguments. In this case, the last read element will be the one corresponding to the value of such arguments. Table can be read backward by assigning a negative *kloop* value.

*trigseq* is designed to be used together with *seqtime* or *trigger* opcodes.

## See Also

*seqtime, trigger*

## Credits

Author: Gabriel Maldonado

November 2002. Added a note about the *kinitndx* parameter, thanks to Rasmus Ekman.

January 2003. Thanks to a note from Øyvind Brandtsegg, I corrected the credits.

New in version 4.06

# trirand

trirand — Générateur de nombres aléatoires de distribution triangulaire.

## Description

Générateur de nombres aléatoires de distribution triangulaire. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares trirand krange
```

```
ires trirand krange
```

```
kres trirand krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (*-krange* à *+krange*).

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode trirand. Il utilise le fichier *trirand.csd* [examples/trirand.csd].

### Exemple 539. Exemple de l'opcode trirand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o trirand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```



```
instr 1
  ; Generate a random number between -1 and 1.
  ; krange = 1

  il trirand 1

  print il
endin

; Instrument #2.
instr 2
  ; Generate a random number between -1 and 1.
  ; krange = 1

  seed 0

  il trirand 1

  print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  il = 7506.261
```

## Voir Aussi

*betarand, bexprnd, cauchy, exprand, gauss, linrand, pcauchy, poisson, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.

# trlowest

trlowest — Extracts the lowest-frequency track from a streaming track input signal.

## Description

The trlowest opcode takes an input containing TRACKS pv streaming signals (as generated, for instance by partials) and outputs only the lowest track. In addition it outputs two k-rate signals, corresponding to the frequency and amplitude of the lowest track signal.

## Syntax

```
fsig, kfr,kamp trlowest finl, kscal
```

## Performance

*fsig* -- output pv stream in TRACKS format

*kfr* -- frequency (in Hz) of the lowest-frequency track

*kamp* -- amplitude of the lowest-frequency track

*fin* -- input pv stream in TRACKS format.

*kscal* -- amplitude scaling of output.

## Examples

### Exemple 540. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
flow,kfr,kamp trlowest fst,1 ; lowest freq-track
aout tradsyn flow, 1, 1, 1, 1 ; resynthesis of lowest frequency
out aout
```

The example above shows partial tracking of an ifd-analysis signal, extraction of the lowest frequency and resynthesis.

## Credits

Author: Victor Lazzarini  
February 2006

New in Csound5.01

# trmix

trmix — Streaming partial track mixing.

## Description

The `trmix` opcode takes two inputs containing TRACKS pv streaming signals (as generated, for instance by `partials`) and mixes them into a single TRACKS stream. Tracks will be mixed up to the available space (defined by the original number of FFT bins in the analysed signals). If the sum of the input tracks exceeds this space, the higher-ordered tracks in the second input will be pruned.

## Syntax

```
fsig trmix fin1, fin2
```

## Performance

*fsig* -- output pv stream in TRACKS format

*fin1* -- first input pv stream in TRACKS format.

*fin2* -- second input pv stream in TRACKS format

## Examples

### Exemple 541. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
fslo,fshi trsplit fst, 1500 ; split partial tracks at 1500 Hz
fscl trscale fshi, 1.15 ; shift the upper tracks
fmix trmix fslo,fscl ; mix the shifted and unshifted tracks
aout trdsyn fmix, 1, 1, 500, 1 ; resynthesis of tracks
out aout
```

The example above shows partial tracking of an ifd-analysis signal, frequency splitting and pitch shifting of the upper part of the spectrum, followed by the remix of the two parts of the spectrum and resynthesis.

## Credits

Author: Victor Lazzarini  
February 2006

New in Csound5.01

# trscale

trscale — Streaming partial track frequency scaling.

## Description

The `trscale` opcode takes an input containing a TRACKS pv streaming signal (as generated, for instance by `partials`) and scales all frequencies by a k-rate amount. It can also, optionally, scale the gain of the signal by a k-rate amount (default 1). The result is pitch shifting of the input tracks.

## Syntax

```
fsig trscale fin, kpitch[, kgain]
```

## Performance

*fsig* -- output pv stream in TRACKS format

*fin* -- input pv stream in TRACKS format

*kpitch* -- frequency scaling

*kgain* -- amplitude scaling (default 1)

## Examples

### Exemple 542. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
fsc1 trscale fst, 1.5 ; frequency scale (up a 5th)
      aout tradsyn fsc1, 1, 1, 500, 1 ; resynthesis
out aout
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis with pitch shifting.

## Credits

Author: Victor Lazzarini  
February 2006

New in Csound5.01

# trshift

trshift — Streaming partial track frequency scaling.

## Description

The trshift opcode takes an input containing a TRACKS pv streaming signal (as generated, for instance by partials) and shifts all frequencies by a k-rate frequency. It can also, optionally, scale the gain of the signal by a k-rate amount (default 1). The result is frequency shifting of the input tracks.

## Syntax

```
fsig trshift fin, kpsift[, kgain]
```

## Performance

*fsig* -- output pv stream in TRACKS format

*fin* -- input pv stream in TRACKS format

*kshift* -- frequency shift in Hz

*kgain* -- amplitude scaling (default 1)

## Examples

### Exemple 543. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
fsc1 trshift fst, 150 ; frequency shift (adds 150Hz to all tracks)
      aout tradsyn fsc1, 1, 1, 500, 1 ; resynthesis
out aout
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis with frequency shifting.

## Credits

Author: Victor Lazzarini  
February 2006

New in Csound5.01

# trsplit

trsplit — Streaming partial track frequency splitting.

## Description

The trsplit opcode takes an input containing a TRACKS pv streaming signal (as generated, for instance by partials) and splits it into two signals according to a k-rate frequency 'split point'. The first output will contain all tracks up from 0Hz to the split frequency and the second will contain the tracks from the split frequency up to the Nyquist. It can also, optionally, scale the gain of the output signals by a k-rate amount (default 1). The result is two output signals containing only part of the original spectrum.

## Syntax

```
fsiglow, fsighi trsplit fin, ksplit[, kgainlow, kgainhigh]
```

## Performance

*fsiglow* -- output pv stream in TRACKS format containing the tracks below the split point.

*fsighi* -- output pv stream in TRACKS format containing the tracks above and including the split point.

*fin* -- input pv stream in TRACKS format

*ksplit* -- frequency split point in Hz

*kgainlow, kgainhig* -- amplitude scaling of each one of the outputs (default 1).

## Examples

### Exemple 544. Example

```
ain inch 1 ; input signal
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
fslo,fshi trsplit fst, 1500 ; split partial tracks at 1500 Hz
aout tradsyn fshi, 1, 1, 500, 1 ; resynthesis of tracks above 1500Hz
out aout
```

The example above shows partial tracking of an ifd-analysis signal and linear additive resynthesis of the upper part of the spectrum (from 1500Hz).

## Credits

Author: Victor Lazzarini  
February 2006

New in Csound5.01

# turnoff

turnoff — Permet à un instrument de s'arrêter lui-même.

## Description

Permet à un instrument de s'arrêter lui-même.

## Syntaxe

`turnoff`

## Exécution

*turnoff* -- cette instruction de la phase d'exécution permet à un instrument de s'arrêter lui-même. Quelle soit de durée finie ou « tenue », la note en cours d'exécution par l'instrument est immédiatement enlevée de la liste des notes actives. Aucune autre note n'est affectée.

## Exemples

L'exemple suivant utilise l'opcode `turnoff`. Il provoque la fin d'une note lorsqu'un signal de contrôle dépasse un certain seuil (ici la fréquence de Nyquist). Il utilise le fichier *turnoff.csd* [examples/turnoff.csd].

### Exemple 545. Exemple de l'opcode `turnoff`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o turnoff.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  k1 expon 440, p3/10,880      ; begin gliss and continue
  if k1 < sr/2 kgoto contin    ; until Nyquist detected
    turnoff ; then quit

contin:
  a1 oscil 10000, k1, 1
  out a1
endin

</CsInstruments>
<CsScore>
```



```
; Table #1: an ordinary sine wave.  
f 1 0 32768 10 1  
  
; Play Instrument #1 for 4 seconds.  
i 1 0 4  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*ihold*

# turnoff2

turnoff2 — Arrête une ou des instances d'autres instruments pendant la phase d'exécution.

## Description

Arrête une ou des instances d'autres instruments pendant la phase d'exécution.

## Syntaxe

```
turnoff2 kinsno, kmode, krelease
```

## Exécution

*kinsno* -- instrument à arrêter (peut-être fractionnaire). S'il vaut zéro ou est négatif, aucun instrument n'est arrêté.

*kmode* -- somme des valeurs suivantes :

- 0, 1, ou 2 : arrête toutes les instances (0), seulement les plus anciennes (1), ou seulement les plus récentes (2)
- 4 : n'arrête que les notes dont la partie fractionnaire du numéro d'instrument correspond à *kinsno*, plutôt que d'ignorer la partie fractionnaire.
- 8 : n'arrête que les notes dont la durée est indéfinie ( $p3 < 0$  ou MIDI).

*krelease* -- s'il est non nul, les instances arrêtées peuvent avoir une période d'extinction (release), sinon elles sont désactivées immédiatement (avec possible émission de clics).

## Voir Aussi

*turnoff*

## Crédits

Auteur : Istvan Varga  
2005

Nouveau dans Csound 5.00

# turnon

turnon — Active un instrument pour une durée indéfinie.

## Description

Active un instrument pour une durée indéfinie.

## Syntaxe

```
turnon insnum [, itime]
```

## Initialisation

*insnum* -- numéro de l'instrument à activer

*itime* (facultatif, 0 par défaut) -- délai, en secondes, après lequel l'instrument *insnum* sera activé. Vaut 0 par défaut.

## Exécution

*turnon* active l'instrument *insnum* après un délai de *itime* secondes, ou immédiatement si *itime* n'est pas spécifié. L'instrument reste actif jusqu'à ce qu'il soit explicitement arrêté. (Voir *turnoff*)

# unirand

unirand — Générateur de nombres aléatoires de distribution uniforme (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution uniforme (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares unirand krange
```

```
ires unirand krange
```

```
kres unirand krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (0 - *krange*).

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode unirand. Il utilise le fichier *unirand.csd* [examples/unirand.csd].

### Exemple 546. Exemple de l'opcode unirand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o unirand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; Generate a random number between 0 and 1.
  ; krange = 1

  i1 unirand 1

  print i1
endin

; Instrument #2.
instr 2
  ; Generate a random number between 0 and 1.
  ; krange = 1

  seed 0

  i1 unirand 1

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra une ligne comme celle-ci :

```
instr 1: i1 = 0.840
```

## Voir Aussi

*seed, betarand, bexprnd, cauchy, exprand, gauss, linrand, pcauchy, poisson, trirand, weibull*

## Crédits

Auteur: Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.

# upsamp

upsamp — Modify a signal by up-sampling.

## Description

Modify a signal by up-sampling.

## Syntax

ares **upsamp** ksig

## Performance

*upsamp* converts a control signal to an audio signal. It does it by simple repetition of the kval. *upsamp* is a slightly more efficient form of the assignment, *asig = ksig*.

## Examples

```
asrc buzz      10000,440,20, 1    ; band-limited pulse train
adif diff      asrc              ; emphasize the highs
anew balance   adif, asrc        ; but retain the power
agate reson    asrc,0,440        ; use a lowpass of the original
asamp samphold anew, agate       ; to gate the new audiosig
aout tone      asamp,100         ; smooth out the rough edges
```

## See Also

*diff, downsamp, integ, interp, samphold*

# urd

urd — Un générateur de nombres aléatoires de distribution discrète définie par l'utilisateur que l'on peut utiliser comme une fonction.

## Description

Un générateur de nombres aléatoires de distribution discrète définie par l'utilisateur que l'on peut utiliser comme une fonction.

## Syntaxe

```
aout = urd(ktableNum)
```

```
iout = urd(itableNum)
```

```
kout = urd(ktableNum)
```

## Initialisation

*itableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

## Exécution

*ktableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

*urd* est le même opcode que *dusernd*, mais on peut l'utiliser à la manière d'une fonction.

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Voir Aussi

*cusernd*, *dusernd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16

# vadd

vadd — Adds a scalar value to a vector in a table.

## Description

Adds a scalar value to a vector in a table.

## Syntax

```
vadd ifn, kval, kelements [, kdstoffset] [, kverbose]
```

## Initialization

*ifn* - number of the table hosting the vectorial signal to be processed

## Performance

*kval* - scalar value to be added

*kelements* - number of elements of the vector

*kdstoffset* - index offset for the destination table (Optional, default = 0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vadd* adds the value of *kval* to each element of the vector contained in the table *ifn*, starting from table index *kdstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Note that this opcode runs at k-rate so the value of *kval* is added every control period. Use with care or you will end up with very large numbers (or use *vadd\_i*).

These opcodes (*vadd*, *vmult*, *vpow* and *vexp*) perform numeric operations between a vectorial control signal (hosted by the table *ifn*), and a scalar signal (*kval*). Result is a new vector that overrides old values of *ifn*. All these opcodes work at k-rate.

Negative values for *kdstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the initial vectors. If you want to keep initial vector, use *vcopy* or *vcopy\_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.



### Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *iele-*



ments is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

## Examples

Here is an example of the vadd opcode. It uses the file *vadd.csd* [examples/vadd.csd].

### Exemple 547. Example of the vadd opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vadd ifn1, ival, ielements, idstoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 5 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 8 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1 10 12
i2 1.6 0.2 1
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## See also

*vadd\_i*, *vmult*, *vpow* and *vexp*.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vadd\_i

vadd\_i — Adds a scalar value to a vector in a table.

## Description

Adds a scalar value to a vector in a table.

## Syntax

```
vadd_i ifn, ival, ielements [, idstoffset]
```

## Initialization

*ifn* - number of the table hosting the vectorial signal to be processed

*ielements* - number of elements of the vector

*ival* - scalar value to be added

*idstoffset* - index offset for the destination table

## Performance

*vadd\_i* adds the value of *ival* to each element of the vector contained in the table *ifn*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

This opcode runs only on initialization, there is a k-rate version of this opcode called *vadd*.

Negative values for *idstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the initial vectors. If you want to keep initial vector, use *vcopy* or *vcopy\_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

## Examples

Here is an example of the *vadd\_i* opcode. It uses the file *vadd\_i.csd* [examples/vadd\_i.csd].

### Exemple 548. Example of the vadd\_i opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in
-odac        -iadc        ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vadd_i ifn1, ival, ielements, idstoffset
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
    turnoff
endif

kcount = kcount + 1
        endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>

```

## See also

*vadd*, *vmult\_i*, *vpow\_i* and *vexp\_i*.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vaddv

vaddv — Performs addition between two vectorial control signals

## Description

Performs addition between two vectorial control signals

## Syntax

```
vaddv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

## Performance

*kelements* - number of elements of the two vectors

*kdstoffset* - index offset for the destination (ifn1) table (Default=0)

*ksrcoffset* - index offset for the source (ifn2) table (Default=0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vaddv* adds two vectorial control signals, that is, each element of the first vector is processed (only) with the corresponding element of the other vector. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_iopcode* to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector will not be changed for these elements).



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

Please note that using the same table as source and destination table, might produce unexpected behavior so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are added). There's an i-rate ver-

sion of this opcode called *vaddv\_i*.



## Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

All these operators (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Examples

Here is an example of the *vaddv* opcode. It uses the file *vaddv.csd* [examples/vaddv.csd].

### Exemple 549. Example of the *vaddv* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vaddv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
```

```
        endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16
f 2 0 16 -7 1 15 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vaddv\_i

vaddv\_i — Performs addition between two vectorial control signals at init time.

## Description

Performs addition between two vectorial control signals at init time.

## Syntax

```
vaddv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

*ielements* - number of elements of the two vectors

*idstoffset* - index offset for the destination (*ifn1*) table (Default=0)

*isrcoffset* - index offset for the source (*ifn2*) table (Default=0)

## Performance

*vaddv\_i* adds two vectorial control signals, that is, each element of the first vector is processed (only) with the corresponding element of the other vector. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector will not be changed for these elements).



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vaddv*.

All these operators (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.



## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vaget

vaget — Access values of the current buffer of an a-rate variable by indexing.

## Description

Access values of the current buffer of an a-rate variable by indexing. Useful for doing sample-by-sample manipulation at k-rate without using setksmps 1.



### Note

Because this opcode does not do any bounds checking, the user must be careful not to try to read values past ksmpls (the size of a buffer for an a-rate variable) by using index values greater than ksmpls.

## Syntax

```
kval vaget kndx, avar
```

## Performance

*kval* - value read from avar

*kndx* - index of the sample to read from the current buffer of the given avar variable

*avar* - a-rate variable to read from

## Examples

Here is an example of the vaget opcode. It uses the file *vaget.csd* [examples/vaget.csd].

### Exemple 550. Example of the vaget opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o avarget.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=44100
ksmps=16
nchnls=2

instr 1 ; Sqrt Signal
ifreq = (p4 > 15 ? p4 : cpspch(p4))
iamp = ampdb(p5)

aout init 0
```

```
ksampnum init 0
kenv linseg 0, p3 * .5, 1, p3 * .5, 0

aout1 vco2 1, ifreq
aout2 vco2 .5, ifreq * 2
aout3 vco2 .2, ifreq * 4

aout sum          aout1, aout2, aout3

;Take Sqrt of signal, checking for negatives
kcount = 0

loopStart:

    kval vaget kcount,aout

    if (kval > .0) then
        kval = sqrt(kval)
    elseif (kval < 0) then
        kval = sqrt(-kval) * -1
    else
        kval = 0
    endif

    vaset kval, kcount,aout

loop_lt kcount, 1, ksmps, loopStart

aout = aout * kenv

aout moogladder aout, 8000, .1

aout = aout * iamp

outs aout, aout
endin

</CsInstruments>
<CsScore>

i1 0.0 2 440 80
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*vaset*

## Credits

Author: Steven Yi

New in version 5.04

September 2006.

# valpass

valpass — Variably reverberates an input signal with a flat frequency response.

## Description

Variably reverberates an input signal with a flat frequency response.

## Syntax

```
ares valpass asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]
```

## Initialization

*imaxlpt* -- maximum loop time for *klpt*

*iskip* (optional, default=0) -- initial disposition of delay-loop data space (cf. *reson*). The default value is 0.

*insmps* (optional, default=0) -- delay amount, as a number of samples.

## Performance

*krvt* -- the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude).

*xlpt* -- variable loop time in seconds, same as *ilpt* in *comb*. Loop time can be as large as *imaxlpt*.

This filter reiterates input with an echo density determined by loop time *ilpt*. The attenuation rate is independent and is determined by *krvt*, the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude). Its output will begin to appear immediately.

## See Also

*alpass*, *comb*, *reverb*, *vcomb*

## Credits

Author: William « Pete » Moss  
University of Texas at Austin  
Austin, Texas USA  
January 2002

# vaset

vaset — Write value of into the current buffer of an a-rate variable by index.

## Description

Write values into the current buffer of an a-rate variable at the given index. Useful for doing sample-by-sample manipulation at k-rate without using setksmps 1.



### Note

Because this opcode does not do any bounds checking, the user must be careful not to try to write values past ksmpls (the size of a buffer for an a-rate variable) by using index values greater than ksmpls.

## Syntax

```
vaset kval, kndx, avar
```

## Performance

*kval* - value to write into avar

*kndx* - index of the sample to write to the current buffer of the given avar variable

*avar* - a-rate variable to write to

## Examples

Here is an example of the vaset opcode. It uses the file *vaset.csd* [examples/vaset.csd].

### Exemple 551. Example of the vaset opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o avarset.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=44100
ksmps=1
nchnls=2

instr 1 ; Sine Wave
ifreq = (p4 > 15 ? p4 : cpspch(p4))
iamp = ampdb(p5)

kenv adsr 0.1, 0.05, .9, 0.2
```

```
aout init 0
ksampnum init 0

kcount = 0

iperiod = sr / ifreq
i2pi = 3.14159 * 2

loopStart:

kphase = (ksampnum % iperiod) / iperiod
knewval = sin(kphase * i2pi)
    vaset knewval, kcount, aout
    ksampnum = ksampnum + 1

loop_lt kcount, 1, ksmps, loopStart

aout = aout * iamp * kenv
outs aout, aout
    endin

</CsInstruments>
<CsScore>

i1 0.0 2 440 80
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*vaget*

## Credits

Author: Steven Yi

New in version 5.04

September 2006.

# vbap16

vbap16 — Distributes an audio signal among 16 channels.

## Description

Distributes an audio signal among 16 channels.

## Syntax

```
ar1, ..., ar16 vbap16 asig, kazim [, kelev] [, kspread]
```

## Performance

*asig* -- audio signal to be panned

*kazim* -- azimuth angle of the virtual source

*kelev* (optional) -- elevation angle of the virtual source

*kspread* (optional) -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *kspread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

*vbap16* takes an input signal, *asig*, and distribute it among 16 outputs, according to the controls *kazim* and *kelev*, and the configured loudspeaker placement. If *idim* = 2, *kelev* is set to zero. The distribution is performed using Vector Base Amplitude Panning (VBAP - See reference). VBAP distributes the signal using loudspeaker data configured with *vbaplsinit*. The signal is applied to, at most, two loudspeakers in 2-D loudspeaker configurations, and three loudspeakers in 3-D loudspeaker configurations. If the virtual source is panned outside the region spanned by loudspeakers, the nearest loudspeakers are used in panning.



### Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

## Examples

See the entry for *vbap8* for an example of usage of the *vbap* opcodes.

## Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

## Credits

Author: Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratory of Acoustics and Audio Signal Processing  
Helsinki University of Technology  
Helsinki, Finland  
May 2000

New in Csound Version 4.07. Input parameters accept k-rate since Csound 5.09.



# vbap16move

vbap16move — Distribute an audio signal among 16 channels with moving virtual sources.

## Description

Distribute an audio signal among 16 channels with moving virtual sources.

## Syntax

```
ar1, ..., ar16 vbap16move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]
```

## Initialization

*idur* -- the duration over which the movement takes place.

*ispread* -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *ispread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

*ifldnum* -- number of fields (absolute value must be 2 or larger). If *ifldnum* is positive, the virtual source movement is a polyline specified by given directions. Each transition is performed in an equal time interval. If *ifldnum* is negative, specified angular velocities are applied to the virtual source during specified relative time intervals (see below).

*ifld1, ifld2, ...* -- azimuth angles or angular velocities, and relative durations of movement phases.

## Performance

*asig* -- audio signal to be panned

*vbap16move* allows the use of moving virtual sources. If *ifldnum* is positive, the fields represent directions of virtual sources and equal times, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. The position of the virtual source is interpolated between directions starting from the first direction and ending at the last. Each interval is interpolated in time that is fraction  $\text{total\_time} / \text{number\_of\_intervals}$  of the duration of the sound event.

If *ifldnum* is negative, the fields represent angular velocities and equal times. The first field is, however, the starting direction, *iazi1*, [*iele1*,] *iazi\_vel1*, [*iele\_vel1*,] *iazi\_vel2*, [*iele\_vel2*,] .... Each velocity is applied to the note that is fraction  $\text{total\_time} / \text{number\_of\_velocities}$  of the duration of the sound event. If the elevation of the virtual source becomes greater than 90 degrees or less than 0 degrees, the polarity of angular velocity is changed. Thus the elevational angular velocity produces a virtual source that moves up and down between 0 and 90 degrees.



### Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

## Examples

See the entry for *vbap8move* for an example of usage of the *vbapXmove* opcodes.

## Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*, *vbapzmove*

## Credits

Author: Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratory of Acoustics and Audio Signal Processing  
Helsinki University of Technology  
Helsinki, Finland  
May 2000

New in Csound Version 4.07

# vbap4

vbap4 — Distributes an audio signal among 4 channels.

## Description

Distributes an audio signal among 4 channels.

## Syntax

```
ar1, ar2, ar3, ar4 vbap4 asig, kazim [, kelev] [, kspread]
```

## Performance

*asig* -- audio signal to be panned

*kazim* -- azimuth angle of the virtual source

*kelev* (optional) -- elevation angle of the virtual source

*kspread* (optional) -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *kspread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

*vbap4* takes an input signal, *asig* and distributes it among 4 outputs, according to the controls *kazim* and *kelev*, and the configured loudspeaker placement. If *idim* = 2, *kelev* is set to zero. The distribution is performed using Vector Base Amplitude Panning (VBAP - See reference). VBAP distributes the signal using loudspeaker data configured with *vbaplsinit*. The signal is applied to, at most, two loudspeakers in 2-D loudspeaker configurations, and three loudspeakers in 3-D loudspeaker configurations. If the virtual source is panned outside the region spanned by loudspeakers, the nearest loudspeakers are used in panning.



### Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

## Examples

See the entry for *vbap8* for an example of usage of the *vbap* opcodes.

## Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16*, *vbap16move*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

## Credits

Author: Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratory of Acoustics and Audio Signal Processing  
Helsinki University of Technology  
Helsinki, Finland  
May 2000

New in Csound Version 4.06. Input parameters accept k-rate since Csound 5.09.

# vbap4move

vbap4move — Distributes an audio signal among 4 channels with moving virtual sources.

## Description

Distributes an audio signal among 4 channels with moving virtual sources.

## Syntax

```
ar1, ar2, ar3, ar4 vbap4move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]
```

## Initialization

*idur* -- the duration over which the movement takes place.

*ispread* -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *ispread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

*ifldnum* -- number of fields (absolute value must be 2 or larger). If *ifldnum* is positive, the virtual source movement is a polyline specified by given directions. Each transition is performed in an equal time interval. If *ifldnum* is negative, specified angular velocities are applied to the virtual source during specified relative time intervals (see below).

*ifld1, ifld2, ...* -- azimuth angles or angular velocities, and relative durations of movement phases (see below).

## Performance

*asig* -- audio signal to be panned

*vbap4move* allows the use of moving virtual sources. If *ifldnum* is positive, the fields represent directions of virtual sources and equal times, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. The position of the virtual source is interpolated between directions starting from the first direction and ending at the last. Each interval is interpolated in time that is fraction  $\text{total\_time} / \text{number\_of\_intervals}$  of the duration of the sound event.

If *ifldnum* is negative, the fields represent angular velocities and equal times. The first field is, however, the starting direction, *iazi1*, [*iele1*,] *iazi\_vel1*, [*iele\_vel1*,] *iazi\_vel2*, [*iele\_vel2*,] .... Each velocity is applied to the note that is fraction  $\text{total\_time} / \text{number\_of\_velocities}$  of the duration of the sound event. If the elevation of the virtual source becomes greater than 90 degrees or less than 0 degrees, the polarity of angular velocity is changed. Thus the elevational angular velocity produces a virtual source that moves up and down between 0 and 90 degrees.



### Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

## Examples

See the entry for *vbap8move* for an example of usage of the *vbap* opcodes.

## Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16*, *vbap16move*, *vbap4*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

## Credits

Author: Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratory of Acoustics and Audio Signal Processing  
Helsinki University of Technology  
Helsinki, Finland  
May 2000

New in Csound Version 4.07

# vbap8

vbap8 — Distributes an audio signal among 8 channels.

## Description

Distributes an audio signal among 8 channels.

## Syntax

```
ar1, ..., ar8 vbap8 asig, kazim [, kelev] [, kspread]
```

## Performance

*asig* -- audio signal to be panned

*kazim* -- azimuth angle of the virtual source

*kelev* (optional) -- elevation angle of the virtual source

*kspread* (optional) -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *kspread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

*vbap8* takes an input signal, *asig*, and distributes it among 8 outputs, according to the controls *kazim* and *kelev*, and the configured loudspeaker placement. If *idim* = 2, *kelev* is set to zero. The distribution is performed using Vector Base Amplitude Panning (VBAP - See reference). VBAP distributes the signal using loudspeaker data configured with *vbaplsinit*. The signal is applied to, at most, two loudspeakers in 2-D loudspeaker configurations, and three loudspeakers in 3-D loudspeaker configurations. If the virtual source is panned outside the region spanned by loudspeakers, the nearest loudspeakers are used in panning.



### Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

## Example

Here is a simple example of the *vbap8* opcode. It uses the file *vbap8.csd* [examples/vbap8.csd].

### Exemple 552. Example of the vbap8 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
;-odac      -iadc      ;;RT audio I/O
```

```

; For Non-realtime ouput leave only the line below:
-o vbap8.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

    sr          =          41000
    kr          =          441
    ksmpts      =          100
    nchnls      =          4
    vbaplsinit  2, 8,  0, 45, 90, 135, 200, 245, 290, 315

    instr 1
    asig      oscil      20000, 440, 1
    a1,a2,a3,a4,a5,a6,a7,a8  vbap8  asig, p4, 0, 20 ;p4 = azimuth

    ;render twice with alternate outq statements
    ; to obtain two 4 channel .wav files:

        outq      a1,a2,a3,a4
        outq      a5,a6,a7,a8
    ; or use an 8-channel output for realtime output (set nchnls to 8):
    ;      outo a1,a2,a3,a4,a5,a6,a7,a8
    endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
,      azimuth
i 1 0 1      20
i 1 + .      40
i 1 + .      60
i 1 + .      80
i 1 + .     100
i 1 + .     120
i 1 + .     140
i 1 + .     160
e

</CsScore>
</CsoundSynthesizer>

```

## Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16, vbap16move, vbap4, vbap4move, vbap8move, vbaplsinit, vbapz, vbapzmove*

## Credits

Author: Ville Pulkki  
 Sibelius Academy Computer Music Studio  
 Laboratory of Acoustics and Audio Signal Processing  
 Helsinki University of Technology  
 Helsinki, Finland  
 May 2000

New in Csound Version 4.07. Input parameters accept k-rate since Csound 5.09.



# vbap8move

vbap8move — Distributes an audio signal among 8 channels with moving virtual sources.

## Description

Distributes an audio signal among 8 channels with moving virtual sources.

## Syntax

```
ar1, ..., ar8 vbap8move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]
```

## Initialization

*idur* -- the duration over which the movement takes place.

*ispread* -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *ispread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

*ifldnum* -- number of fields (absolute value must be 2 or larger). If *ifldnum* is positive, the virtual source movement is a polyline specified by given directions. Each transition is performed in an equal time interval. If *ifldnum* is negative, specified angular velocities are applied to the virtual source during specified relative time intervals (see below).

*ifld1*, *ifld2*, ... -- azimuth angles or angular velocities, and relative durations of movement phases (see below).

## Performance

*asig* -- audio signal to be panned

*vbap8move* allows the use of moving virtual sources. If *ifldnum* is positive, the fields represent directions of virtual sources and equal times, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. The position of the virtual source is interpolated between directions starting from the first direction and ending at the last. Each interval is interpolated in time that is fraction  $\text{total\_time} / \text{number\_of\_intervals}$  of the duration of the sound event.

If *ifldnum* is negative, the fields represent angular velocities and equal times. The first field is, however, the starting direction, *iazi1*, [*iele1*,] *iazi\_vel1*, [*iele\_vel1*,] *iazi\_vel2*, [*iele\_vel2*,] .... Each velocity is applied to the note that is fraction  $\text{total\_time} / \text{number\_of\_velocities}$  of the duration of the sound event. If the elevation of the virtual source becomes greater than 90 degrees or less than 0 degrees, the polarity of angular velocity is changed. Thus the elevational angular velocity produces a virtual source that moves up and down between 0 and 90 degrees.



### Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

## Example

Here is a simple example of the *vbap8move* opcode. It uses the file *vbap8move.csd* [examples/vbap8move.csd].

### Exemple 553. Example of the *vbap8move* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc             ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vbap4move.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 10
nchnls = 8

;Example by Hector Centeno 2007

vbaplsinit      2, 8, 15, 65, 115, 165, 195, 245, 295, 345

    instr 1
ifldnum = 9
ispread = 30
idur = p3

;; Generate a sound source
kenv loopseg 10, 0, 0, 0, 0.5, 1, 10, 0
al pinkish 3000*kenv

;; Move circling around once all the speakers
aout1, aout2, aout3, aout4, aout5, aout6, aout7, aout8 vbap8move al, idur, ispread, ifldnum, 15, 65, 115, 165, 195, 245, 295, 345

;; Speaker mapping
aFL = aout8 ; Front Left
aFR = aout1 ; Front Right
aMFL = aout7 ; Mid Front Left
aMFR = aout2 ; Mid Front Right
aMBL = aout6 ; Mid Back Left
aMBR = aout3 ; Mid Back Right
aBL = aout5 ; Back Left
aBR = aout4 ; Back Right

outo aFL, aFR, aMFL, aMFR, aMBL, aMBR, aBL, aBR

    endin

</CsInstruments>
<CsScore>
i1 0 30
e
</CsScore>
</CsoundSynthesizer>
```

## Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16, vbap16move, vbap4, vbap4move, vbap8, vbaplsinit, vbapz, vbapzmove*

## Credits

Author: Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratory of Acoustics and Audio Signal Processing  
Helsinki University of Technology  
Helsinki, Finland  
May 2000

New in Csound Version 4.07

# vbaplsinit

vbaplsinit — Configures VBAP output according to loudspeaker parameters.

## Description

Configures VBAP output according to loudspeaker parameters.

## Syntax

```
vbaplsinit idim, ilsnum [, idir1] [, idir2] [...] [, idir32]
```

## Initialization

*idim* -- dimensionality of loudspeaker array. Either 2 or 3.

*ilsnum* -- number of loudspeakers. In two dimensions, the number can vary from 2 to 16. In three dimensions, the number can vary from 3 and 16.

*idir1*, *idir2*, ..., *idir32* -- directions of loudspeakers. Number of directions must be less than or equal to 16. In two-dimensional loudspeaker positioning, *idirn* is the azimuth angle respective to *n*th channel. In three-dimensional loudspeaker positioning, fields are the azimuth and elevation angles of each loudspeaker consequently (*azi1*, *ele1*, *azi2*, *ele2*, etc.).

## Performance

VBAP distributes the signal using loudspeaker data configured with *vbaplsinit*. The signal is applied to, at most, two loudspeakers in 2-D loudspeaker configurations, and three loudspeakers in 3-D loudspeaker configurations. If the virtual source is panned outside the region spanned by loudspeakers, the nearest loudspeakers are used in panning.

## Examples

See the entry for *vbap16move* and *vbap8* for examples of usage of the *vbap* opcodes.

## Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16*, *vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbapz*, *vbapzmove*

## Credits

Author: Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratory of Acoustics and Audio Signal Processing

Helsinki University of Technology  
Helsinki, Finland  
May 2000

New in Csound Version 4.07

# vbapz

vbapz — Writes a multi-channel audio signal to a ZAK array.

## Description

Writes a multi-channel audio signal to a ZAK array.

## Syntax

```
vbapz inumchnls, istartndx, asig, kazim [, kelev] [, kspread]
```

## Initialization

*inumchnls* -- number of channels to write to the ZA array. Must be in the range 2 - 256.

*istartndx* -- first index or position in the ZA array to use

## Performance

*asig* -- audio signal to be panned

*kazim* -- azimuth angle of the virtual source

*kelev* (optional) -- elevation angle of the virtual source

*kspread* (optional) -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *kspread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

The opcode *vbapz* is the multiple channel analog of the opcodes like *vbap4*, working on *inumchnls* and using a ZAK array for output.



### Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

## Examples

See the entry for *vbap8* for an example of usage of the *vbap* opcodes.

## Reference

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16*, *vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapzmove*

## Credits

John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
May 2000

New in Csound Version 4.07. Input parameters accept k-rate since Csound 5.09.

# vbapzmove

vbapzmove — Writes a multi-channel audio signal to a ZAK array with moving virtual sources.

## Description

Writes a multi-channel audio signal to a ZAK array with moving virtual sources.

## Syntax

```
vbapzmove inumchnls, istartndx, asig, idur, ispread, ifldnum, ifld1, \
        ifld2, [...]
```

## Initialization

*inumchnls* -- number of channels to write to the ZA array. Must be in the range 2 - 256.

*istartndx* -- first index or position in the ZA array to use

*idur* -- the duration over which the movement takes place.

*ispread* -- spreading of the virtual source (range 0 - 100). If value is zero, conventional amplitude panning is used. When *ispread* is increased, the number of loudspeakers used in panning increases. If value is 100, the sound is applied to all loudspeakers.

*ifldnum* -- number of fields (absolute value must be 2 or larger). If *ifldnum* is positive, the virtual source movement is a polyline specified by given directions. Each transition is performed in an equal time interval. If *ifldnum* is negative, specified angular velocities are applied to the virtual source during specified relative time intervals (see below).

*ifld1*, *ifld2*, ... -- azimuth angles or angular velocities, and relative durations of movement phases (see below).

## Performance

*asig* -- audio signal to be panned

The opcode *vbapzmove* is the multiple channel analog of the opcodes like *vbap4move*, working on *inumchnls* and using a ZAK array for output.



### Avertissement

Please note that all *vbap* panning opcodes require the *vbap* system to be initialized using *vbaplsinit*.

## Examples

See the entry for *vbap8move* for an example of usage of the *vbap* opcodes.

## Reference



Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, 1997 June, Vol. 45/6, p. 456.

## See Also

*vbap16, vbap16move, vbap4, vbap4move, vbap8, vbap8move, vbaplsinit, vbapz,*

## Credits

John ffitc  
University of Bath/Codemist Ltd.  
Bath, UK  
May 2000

New in Csound Version 4.07

# vcella

vcella — Automate Cellulaire

## Description

Automate Cellulaire unidimensionnel appliqué à des vecteurs de Csound.

## Syntaxe

```
vcella ktrig, kreinit, ioutFunc, initStateFunc, \  
         iRuleFunc, ielements, irulelen [, iradius]
```

## Initialisation

*ioutFunc* - numéro de la table dans laquelle l'état de chaque cellule est stocké

*initStateFunc* - numéro de la table contenant l'état initial de chaque cellule

*iRuleFunc* - numéro de la table de consultation contenant les règles

*ielements* - nombre total de cellules

*irulelen* - nombre total de règles

*iradius* (facultatif) - rayon de l'Automate Cellulaire. Actuellement, le rayon de l'AC peut valoir 1 ou 2 (la valeur par défaut est 1)

## Exécution

*ktrig* - signal de déclenchement. Chaque fois qu'il est non nul, une nouvelle génération de cellules est évaluée.

*kreinit* - signal de déclenchement. Chaque fois qu'il est non nul, l'état de toutes les cellules est forcé à celui de *initStateFunc*.

*vcella* met en œuvre un automate cellulaire pour lequel l'état de chaque cellule est stocké dans *ioutFunc*. Ainsi *ioutFunc* est un vecteur contenant l'état courant de chaque cellule. Ce vecteur variable peut être utilisé avec d'autres opcodes basés sur des vecteurs, tels que *adsynt*, *vmap*, *vpowv* etc.

*initStateFunc* est un vecteur d'entrée contenant la valeur initiale de la rangée de cellules, tandis que *iRuleFunc* est un vecteur d'entrée contenant les règles sous la forme d'une table de consultation. Notez que *initStateFunc* et *iRuleFunc* peuvent être modifiés pendant l'exécution au moyen d'autres opcodes basés sur des vecteurs (par exemple *vcopy*) afin de forcer un changement de règle et d'état pendant l'exécution.

Une nouvelle génération de cellules est évaluée chaque fois que *ktrig* contient une valeur non nulle. De plus, l'état de toutes les cellules peut être forcé à l'état correspondant dans *initStateFunc* chaque fois que *kreinit* contient une valeur non nulle.

Le rayon de l'algorithme d'AC peut valoir 1 ou 2 (argument facultatif *iradius*).

## Exemples

Voici un exemple de l'opcode `vcella`. Il utilise le fichier `vcella.csd` [exemples/vcella.csd].

L'exemple suivant utilise l'opcode `vcella`

### Exemple 554. Exemple de l'opcode `vcella`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vcella.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; vcella.csd
; by Anthony Kozar

; This file demonstrates some of the new opcodes available in
; Csound 5 that come from Gabriel Maldonado's CsoundAV.

sr          = 44100
kr          = 4410
ksmps      = 10
nchnls     = 1

; Cellular automata-driven oscillator bank using vcella and adsynt
instr 1
  idur      = p3
  iCarate   = p4                                ; number of times per second the CA calculates new values

  ; f-tables for CA parameters
  iCAinit   = p5                                ; CA initial states
  iCARule   = p6                                ; CA rule values
  ; The rule is used as follows:
  ; the states (values) of each cell are summed with their neighboring cells within
  ; the specified radius (+/- 1 or 2 cells). Each sum is used as an index to read a
  ; value from the rule table which becomes the new state value for its cell.
  ; All new states are calculated first, then the new values are all applied
  ; simultaneously.

  ielements = ftlen(iCAinit)
  inumrules  = ftlen(iCARule)
  iradius    = 1

  ; create some needed tables
  iCAstate   ftgen 0, 0, ielements, -2, 0      ; will hold the current CA states
  ifreqs     ftgen 0, 0, ielements, -2, 0      ; will hold the oscillator frequency for each cell
  iamps      ftgen 0, 0, ielements, -2, 0      ; will hold the amplitude for each cell

  ; calculate cellular automata state
  ktrig      metro iCarate                      ; trigger the CA to update iCarate times per second
  vcella     ktrig, 0, iCAstate, iCAinit, iCARule, ielements, inumrules, iradius

  ; scale CA state for use as amplitudes of the oscillator bank
  vcopy      iamps, iCAstate, ielements
  vmult      iamps, (1/3), ielements           ; divide by 3 since state values are 0-3

  vport      iamps, .01, ielements             ; need to smooth the amplitude changes for adsynt
  ; we could use adsynt2 instead of adsynt, but it does not seem to be working

  ; i-time loop for calculating frequencies
  index      = 0
  inew       = 1
  iratio     = 1.125                            ; just major second (creating a whole tone scale)
loop1:
  tableiw    inew, index, ifreqs, 0            ; 0 indicates integer indices
  inew       = inew * iratio
  index      = index + 1
  if (index < ielements) igoto loop1
```

```

; create sound with additive oscillator bank
ifreqbase = 64
iwavefn    = 1
iphs       = 2                                ; random oscillator phases

kenv        linseg    0.0, 0.5, 1.0, idur - 1.0, 1.0, 0.5, 0.0
aosc        adsynt    kenv, ifreqbase, iwavefn, ifreqs, iamps, ielements, iphs

                                out        aosc * ampdb(68)
endin

</CsInstruments>
<CsScore>
f1 0 16384 10 1

; This example uses a 4-state cellular automata
; Possible state values are 0, 1, 2, and 3

; CA initial state
; We have 16 cells in our CA, so the initial state table is size 16
f10 0 16 -2 0 1 0 0 1 0 0 2 2 0 0 1 0 0 1 0

; CA rule
; The maximum sum with radius 1 (3 cells) is 9, so we need 10 values in the rule (0-9)
f11 0 16 -2 1 0 3 2 1 0 0 2 1 0

; Here is our one and only note!
i1 0 20 4 10 11

e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par : Gabriel Maldonado.

Nouveau dans Csound 5 (Disponible auparavant seulement dans CsoundAV)

Exemple par : Anthony Kozar

## VCO

vco — Implémentation de la modélisation d'un oscillateur analogique à bande de fréquence limitée.

## Description

Implémentation de la modélisation d'un oscillateur analogique à bande de fréquence limitée, basée sur l'intégration d'impulsions à bande de fréquence limitée. *vco* peut être utilisé pour simuler différentes formes d'onde analogiques.

## Syntaxe

```
ares vco xamp, xcps, iwave, kpw [, ifn] [, imaxd] [, ileak] [, inyx] \  
    [, iphs] [, iskip]
```

## Initialisation

*iwave* -- détermine la forme d'onde :

- *iwave* = 1 - dent de scie
- *iwave* = 2 - carrée/PWM
- *iwave* = 3 - triangle/dent de scie/rampe

*ifn* (facultatif, par défaut 1) -- numéro de table d'une fonction sinus stockée. Doit pointer sur une table valide qui contient une onde sinus. Csound rapportera une erreur si ce paramètre n'est pas fixé et que la table n°1 n'existe pas.

*imaxd* (facultatif, par défaut 1) -- temps de retard maximum. Une durée de  $1/_{ifqc}$  peut être nécessaire pour les formes d'onde PWM et triangle. Le temps d'ajustement de la hauteur à cette valeur peut aller jusqu'à  $1/(\text{fréquence minimale})$ .

*ileak* (facultatif, par défaut 0) -- si *ileak* se situe entre zéro et un ( $0 < ileak < 1$ ), *ileak* est utilisé comme facteur de fuite de l'intégrateur. Sinon un facteur de fuite de 0,999 est utilisé pour les ondes en dent de scie et carrée et de 0,995 pour l'onde triangle. On peut l'utiliser pour « aplatiser » l'onde carrée ou « renforcer » l'onde en dent de scie dans les fréquences basses en fixant *ileak* à 0,99999 ou à une valeur semblable. Le résultat devrait être une onde carrée sonnant plus faux.

*inyx* (facultatif, par défaut 0,5) -- est utilisé pour déterminer le nombre d'harmoniques dans l'impulsion à bande de fréquence limitée. Tous les harmoniques jusqu'à  $sr * inyx$  seront utilisés. La valeur par défaut donne  $sr * 0,5$  ( $sr/2$ ). Pour  $sr/4$  utiliser *inyx* = 0,25. Cela peut générer un son plus « gras » dans certains cas.

*iphs* (facultatif, par défaut 0) -- c'est une valeur de phase. Il y a un artefact (comme un bogue) dans *vco* qui se produit pendant la première demi-période de l'onde carrée et qui rend la forme d'onde plus grande en amplitude que les autres. La valeur de *iphs* a un effet sur cet artefact. En particulier, si l'on fixe *iphs* à 0,5 la première demi-période de l'onde carrée ressemblera à une petite onde triangulaire. Ceci peut être préférable à la grande forme d'onde de l'artefact qui est le comportement par défaut.

*iskip* (facultatif, par défaut 0) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

## Exécution

*kpw* -- détermine soit la largeur de la pulsation (si *iwave* vaut 2) soit le caractère de la dent de scie / rampe (si *iwave* vaut 3). La valeur de *kpw* doit être supérieure à 0 et inférieure à 1. Une valeur de 0,5 génèrera une onde carrée (si *iwave* vaut 2) ou une onde triangle (si *iwave* vaut 3).

*xamp* -- détermine l'amplitude

*xcps* -- fréquence de l'onde en cycles par seconde.

## Exemples

Voici un exemple de l'opcode *vco*. Il utilise le fichier *vco.csd* [examples/vco.csd].

### Exemple 555. Exemple de l'opcode *vco*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vco.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1
instr 1
; Set the amplitude.
kamp = p4

; Set the frequency.
kcps = cpspch(p5)

; Select the wave form.
iwave = p6

; Set the pulse-width/saw-ramp character.
kpw init 0.5

; Use Table #1.
ifn = 1

; Generate the waveform.
asig vco kamp, kcps, iwave, kpw, ifn

; Output and amplification.
out asig
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 65536 10 1

; Define the score.
; p4 = raw amplitude (0-32767)
; p5 = frequency, in pitch-class notation.
```

```
; p6 = the waveform (1=Saw, 2=Square/PWM, 3=Tri/Saw-Ramp-Mod)
i 1 00 02 20000 05.00 1
i 1 02 02 20000 05.00 2
i 1 04 02 20000 05.00 3

i 1 06 02 20000 07.00 1
i 1 08 02 20000 07.00 2
i 1 10 02 20000 07.00 3

i 1 12 02 20000 09.00 1
i 1 14 02 20000 09.00 2
i 1 16 02 20000 09.00 3

i 1 18 02 20000 11.00 1
i 1 20 02 20000 11.00 2
i 1 22 02 20000 11.00 3
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*vco2*

## Crédits

Auteur : Hans Mikelson  
Décembre 1998

Nouveau dans la version 3.50 de Csound

Novembre 2002. Correction de la documentation pour le paramètre *kpw*. Merci à Luis Jure et à Hans Mikelson.

## vco2

*vco2* — Implémentation d'un oscillateur à bande de fréquence limitée qui utilise des tables pré-calculées.

## Description

*vco2* est semblable à *vco*. Mais l'implémentation utilise des tables pré-calculées de formes d'onde à bande de fréquence limitée (voir aussi *GEN30*) plutôt que d'intégrer des impulsions. Cet opcode peut être plus rapide que *vco* (particulièrement lors de l'utilisation d'un faible taux de contrôle) et il permet également une meilleure qualité sonore. De plus, il y a plus de formes d'onde et la phase de l'oscillateur peut être modulée au taux-k. Il a pour inconvénient une utilisation plus importante de la mémoire. Pour plus de détails sur les tables de *vco2*, voir aussi *vco2init* et *vco2ft*.

## Syntaxe

```
ares vco2 kamp, kcps [, imode] [, kpw] [, kphs] [, inyx]
```

## Initialisation

*imode* (facultatif, par défaut 0) -- somme des valeurs représentant la forme d'onde et ses valeurs de contrôle.

On peut utiliser ces valeurs pour *imode* :

- 16 : active le contrôle de la phase au taux-k (s'il est positionné, *kphs* est un paramètre de taux-k nécessaire pour permettre la modulation de la phase)
- 1 : ignorer l'initialisation

On peut utiliser exactement une seule de ces valeurs de *imode* pour choisir la forme d'onde à générer :

- 14 : forme d'onde -1 définie par l'utilisateur (nécessite l'utilisation de l'opcode *vco2init*)
- 12 : triangle (pas de rampe, plus rapide)
- 10 : onde carrée (pas de PWM, plus rapide)
- 8 :  $4 * x * (1 - x)$  (c'est-à-dire l'intégration d'une dent de scie)
- 6 : pulsation (non normalisée)
- 4 : dent de scie / triangle / rampe
- 2 : carrée / PWM
- 0 : dent de scie

La valeur par défaut de *imode* est zéro, ce qui signifie une onde en dent de scie sans contrôle de la phase au taux-k.



*inyx* (facultatif, par défaut 0,5) -- largeur de bande de l'onde générée exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à  $sr/2$ ), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 ( $sr/4$ ), ou à 0,3333 ( $sr/3$ ), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

## Exécution

*ares* -- le signal audio en sortie.

*kamp* -- amplitude. Si *imode* vaut 6 (pulsation), le niveau de sortie réel peut être bien plus élevé que cette valeur.

*kcps* -- fréquence en Hz (doit être dans l'intervalle  $-sr/2$  à  $sr/2$ ).

*kpw* (facultatif) -- largeur de pulsation de l'onde carrée (*imode* = 2) ou caractéristiques de l'onde triangle ou rampe (*imode* = 4). Il n'est requis que pour ces formes d'onde et il est ignoré dans les autres cas. L'intervalle attendu va de 0 à 1, toutes les autres valeurs y étant ramenées cycliquement.



### Avertissement

*kpw* ne doit pas être une valeur entière exacte (0 ou 1) lors de la génération d'une onde en dent de scie / triangle / rampe (*imode* = 4). Dans ce cas, l'intervalle recommandé est d'environ 0,01 à 0,99. Cette limitation n'existe pas pour une forme d'onde carrée/PWM.

*kphs* (facultatif) -- phase de l'oscillateur (en fonction de *imode*, ce sera un paramètre facultatif de taux-i qui vaut zéro par défaut ou un paramètre obligatoire de taux-k). Comme pour *kpw*, l'intervalle attendu va de 0 à 1.



### Note

Si l'on utilise un faible taux de contrôle, la largeur de pulsation (*kpw*) et la modulation de phase (*kphs*) sont converties en interne en modulation de fréquence. Cela permet un traitement plus rapide et réduit le nombre d'artefacts. Mais dans le cas de notes très longues avec des changements rapides et continus de *kpw* ou de *kphs*, la phase peut se décaler par rapport à la valeur voulue. Dans la plupart des cas, l'erreur de phase sera au maximum de 0,037 par heure (en supposant un taux d'échantillonnage de 44100 Hz).

Ceci pose problème principalement avec la largeur d'impulsion (*kpw*) par la possible apparition de divers artefacts. En attendant la résolution de ces problèmes dans de futures versions de *vco2*, les recommandations suivantes peuvent être utiles :

- N'utiliser que des valeurs de *kpw* dans l'intervalle 0,05 à 0,95. (Il y a plus d'artefacts au voisinage des valeurs entières)
- Essayer d'éviter de moduler *kpw* par des formes d'onde asymétriques telles que l'onde en dent de scie. Il est très peu probable qu'une modulation symétrique relativement lente ( $\leq 20$  Hz) (par exemple une onde sinus ou triangle), que des fonctions splines aléatoires (également lentes) ou qu'une pulsation de largeur fixe causent des problèmes de synchronisation.
- Dans certains cas, l'ajout d'un tremblement aléatoire (par exemple, des fonctions spline avec une amplitude d'environ 0,01) à *kpw* peut aussi résoudre le problème.

## Exemples

Voici un exemple de l'opcode vco2. Il utilise le fichier *vco2.csd* [examples/vco2.csd].

### Exemple 556. Exemple de l'opcode vco2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
; -odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vco2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 10
nchnls  = 1

; user defined waveform -1: trapezoid wave with default parameters (can be
; accessed at ftables starting from 10000)
itmp    ftgen 1, 0, 16384, 7, 0, 2048, 1, 4096, 1, 4096, -1, 4096, -1, 2048, 0
ift      vco2init -1, 10000, 0, 0, 0, 1
; user defined waveform -2: fixed table size (4096), number of partials
; multiplier is 1.02 (~238 tables)
itmp    ftgen 2, 0, 16384, 7, 1, 4095, 1, 1, -1, 4095, -1, 1, 0, 8192, 0
ift      vco2init -2, ift, 1.02, 4096, 4096, 2

instr 1
  expon p4, p3, p5                ; instr 1: basic vco2 example
  vco2 12000, kcps                 ; (sawtooth wave with default
  out a1                          ; parameters)
endin

instr 2
  expon p4, p3, p5                ; instr 2:
  linseg 0.1, p3/2, 0.9, p3/2, 0.1 ; PWM example
  vco2 10000, kcps, 2, kpw
  out a1
endin

instr 3
  expon p4, p3, p5                ; instr 3: vco2 with user
  vco2 14000, kcps, 14            ; defined waveform (-1)
  linseg 1, p3 - 0.1, 1, 0.1, 0  ; de-click envelope
  out a1 * aenv
endin

instr 4
  expon p4, p3, p5                ; instr 4: vco2ft example,
  vco2ft kcps, -2, 0.25          ; with user defined waveform
  oscilikt 12000, kcps, kfn      ; (-2), and sr/4 bandwidth
  out a1
endin

</CsInstruments>
<CsScore>

i 1 0 3 20 2000
i 2 4 2 200 400
i 3 7 3 400 20
i 4 11 2 100 200

f 0 14

e

</CsScore>
```

`</CsoundSynthesizer>`

## Voir Aussi

*vco*, *vco2ft*, *vco2ift* et *vco2init*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

## vco2ft

*vco2ft* — Retourne un numéro de table au taux-k pour une fréquence d'oscillateur donnée et une forme d'onde.

## Description

*vco2ft* retourne le numéro d'une table de fonction pour générer la forme d'onde spécifiée à une fréquence donnée. Ce numéro de table de fonction peut être utilisé par n'importe quel opcode de Csound qui génère un signal en lisant une table de fonction (comme *oscilikt*). Les tables doivent avoir été calculées par *vco2init* avant l'appel de *vco2ft* et partagées comme ftables de Csound (*ibasfn*).

## Syntaxe

```
kfn vco2ft kcps, iwave [, inyx]
```

## Initialisation

*iwave* -- la forme d'onde dont le numéro doit être choisi. Les valeurs permises sont :

- 0 : dent de scie
- 1 :  $4 * x * (1 - x)$  (intégration d'une dent de scie)
- 2 : pulsation (non normalisée)
- 3 : onde carrée
- 4 : triangle

De plus, les valeurs négatives de *iwave* sélectionnent des formes d'onde définies par l'utilisateur (voir aussi *vco2init*).

*inyx* (facultatif, par défaut 0,5) -- largeur de bande de la forme d'onde générée, exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à  $sr/2$ ), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 ( $sr/4$ ), ou à 0,3333 ( $sr/3$ ), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

## Exécution

*kfn* -- le numéro de la ftable, retourné au taux-k.

*kcps* -- fréquence en Hz, retournée au taux-k. On peut utiliser zéro ou des valeurs négatives. Cependant, si la valeur absolue dépasse  $sr/2$  (ou  $sr * inyx$ ), la table sélectionnée ne contiendra que du silence.

## Exemples

Voir l'exemple de l'opcode *vco2*.

## Voir Aussi

*vco2ift*, *vco2init* et *vco2*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# vco2ift

*vco2ift* — Retourne un numéro de table au temps-i pour une fréquence d'oscillateur donnée et une forme d'onde.

## Description

*vco2ift* est le même que *vco2ft*, mais il travaille au temps-i. Il est prévu pour être utilisé avec les opcodes qui attendent un numéro de table au taux-i (par exemple, *oscili*).

## Syntaxe

```
ifn vco2ift icps, iwave [, inyx]
```

## Initialisation

*ifn* -- le numéro de ftable.

*icps* -- fréquence en Hz. On peut utiliser zéro ou des valeurs négatives. Cependant, si la valeur absolue dépasse  $sr/2$  (ou  $sr * inyx$ ), la table sélectionnée ne contiendra que du silence.

*iwave* -- la forme d'onde dont le numéro doit être choisi. Les valeurs permises sont :

- 0 : dent de scie
- 1 :  $4 * x * (1 - x)$  (intégration d'une dent de scie)
- 2 : pulsation (non normalisée)
- 3 : onde carrée
- 4 : triangle

De plus, les valeurs négatives de *iwave* sélectionnent des formes d'onde définies par l'utilisateur (voir aussi *vco2init*).

*inyx* (facultatif, par défaut 0,5) -- largeur de bande de la forme d'onde générée, exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à  $sr/2$ ), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 ( $sr/4$ ), ou à 0,3333 ( $sr/3$ ), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

## Voir Aussi

*vco2ft*, *vco2init* et *vco2*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# vco2init

vco2init — Calcul des tables à utiliser par l'opcode *vco2*.

## Description

*vco2init* calcule des tables à utiliser par l'opcode *vco2*. En option, on peut accéder aussi à ces tables comme si elles étaient des tables de fonction standard de Csound. Dans ce cas, on peut utiliser *vco2ft* pour trouver le numéro de table correct pour une fréquence d'oscillateur donnée.

Dans la plupart des cas, cet opcode est appelé depuis l'en-tête de l'orchestre. L'utilisation de *vco2init* dans des instruments est possible mais non recommandée. En effet, le remplacement de tables durant l'exécution peut causer un plantage de Csound si d'autres opcodes sont en train d'accéder à ces tables au même moment.

Notez que *vco2init* n'est pas nécessaire au fonctionnement de *vco2* (les tables sont automatiquement allouées au premier appel de *vco2*, si ce n'est pas déjà fait), cependant il peut être utile dans certains cas :

- Pré-calcul des tables pendant le chargement de l'orchestre. C'est utile lorsque l'on ne veut pas générer les tables pendant l'exécution, afin de ne pas risquer une interruption du traitement en temps réel.
- Partage des tables comme ftables Csound. Par défaut, ces tables ne sont accessibles que par *vco2*.
- Modification des paramètres par défaut des tables (par exemple leur taille) ou utilisation d'une forme d'onde définie par l'utilisateur spécifiée dans une table de fonction.

## Syntaxe

```
ifn vco2init iwave [, ibasfn] [, ipmul] [, iminsiz] [, imaxsiz] [, isrcft]
```

## Initialisation

*ifn* -- le premier numéro de table libre après les tables allouées. Si *ibasfn* n'a pas été spécifié, -1 est retourné.

*iwave* -- somme des valeurs suivantes sélectionnant quelles tables d'onde il faut calculer :

- 16 : triangle
- 8 : onde carrée
- 4 : pulsation (non normalisée)
- 2 :  $4 * x * (1 - x)$  (intégration d'une dent de scie)
- 1 : dent de scie

Alternativement, *iwave* peut être fixé à un entier négatif qui sélectionne une forme d'onde définie par l'utilisateur. Pour cela, le paramètre *isrcft* doit être aussi spécifié. *vco2* peut accéder à la forme d'onde numéro -1. Cependant, les autres formes d'onde définies par l'utilisateur ne sont utilisables qu'avec



*vco2ft* ou *vco2ift*.

*ibasfn* (facultatif, par défaut -1) -- numéro de ftable à partir duquel les opcodes autres que *vco2* peuvent accéder à l'ensemble de tables. Il est nécessaire pour les formes d'onde définies par l'utilisateur, à l'exception de -1. Si cette valeur est inférieure à 1, il n'est pas possible d'accéder aux tables calculées par *vco2init* en tant que tables de fonction de Csound.

*ipmul* (facultatif, par défaut 1,05) -- coefficient multiplicatif pour le nombre d'harmoniques. Si une table a  $n$  harmoniques, la suivante en aura  $n * ipmul$  (au moins  $n + 1$ ). L'intervalle autorisé pour *ipmul* va de 1,01 à 2. Zéro et les valeurs négatives sélectionnent la valeur par défaut (1,05).

*iminsiz* (facultatif, par défaut -1) -- taille de table minimale.

*imaxsiz* (facultatif, par défaut -1) -- taille de table maximale.

La taille de table réelle est calculée en multipliant la racine carrée du nombre d'harmoniques par *iminsiz*, puis en arrondissant le résultat à la puissance de deux supérieure, tout en l'obligeant à ne pas dépasser *imaxsiz*.

Les deux paramètres, *iminsiz* et *imaxsiz*, doivent être des puissances de deux, dans l'intervalle autorisé. L'intervalle autorisé va de 16 à 262144 pour *iminsiz* jusqu'à 16777216 pour *imaxsiz*. Zéro ou des valeurs négatives sélectionnent les réglages par défaut :

- La taille minimale est 128 pour toutes les formes d'onde sauf pour la pulsation (*iwave* = 4). Sa taille minimale est de 256.
- La taille maximale par défaut vaut normalement la taille minimale multipliée par 64, mais pas plus de 16384 si possible. Elle vaut toujours au moins la taille minimale.

*isrcft* (facultatif, par défaut -1) -- numéro de la ftable source pour les formes d'onde définies par l'utilisateur (si *iwave* < 0). *isrcft* doit pointer sur une table de fonction contenant la forme d'onde à utiliser pour générer le tableau de tables. Il est recommandé d'avoir une taille de table d'au moins *imaxsiz* points. Si *iwave* n'est pas négatif (les tables d'onde internes sont utilisées), *isrcft* est ignoré.



## Avertissement

Le nombre et la taille des tables ne sont pas fixes. Les orchestres ne doivent pas dépendre de ces paramètres, car ils peuvent changer d'une version à l'autre de Csound.

Si la table sélectionnée existe déjà, elle est remplacée. Si un opcode est en train d'accéder aux tables au même moment, il est fort probable qu'un plantage se produise. C'est pourquoi il est recommandé de n'utiliser *vco2init* que dans l'en-tête de l'orchestre.

Il ne faut pas remplacer/écraser ces tables par les routines GEN ou l'opcode *ftgen*. Sinon, un comportement imprévisible voire un plantage de Csound peuvent se produire si *vco2* est utilisé. Le premier numéro de ftable libre après le tableau de tables est retourné dans *ifn*.

## Exemples

Voir l'exemple de l'opcode *vco2*.

## Voir Aussi

*vco2ft*, *vco2ift* et *vco2*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# vcomb

vcomb — Variably reverberates an input signal with a « colored » frequency response.

## Description

Variably reverberates an input signal with a « colored » frequency response.

## Syntax

```
ares vcomb asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]
```

## Initialization

*imaxlpt* -- maximum loop time for *klpt*

*iskip* (optional, default=0) -- initial disposition of delay-loop data space (cf. *reson*). The default value is 0.

*insmps* (optional, default=0) -- delay amount, as a number of samples.

## Performance

*krvt* -- the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude).

*xlpt* -- variable loop time in seconds, same as *ilpt* in *comb*. Loop time can be as large as *imaxlpt*.

This filter reiterates input with an echo density determined by loop time *ilpt*. The attenuation rate is independent and is determined by *krvt*, the reverberation time (defined as the time in seconds for a signal to decay to 1/1000, or 60dB down from its original amplitude). Output will appear only after *ilpt* seconds.

## Examples

Here is an example of the vcomb opcode. It uses the file *vcomb.csd* [examples/vcomb.csd].

### Exemple 557. Example of the vcomb opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          -M0 ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

; Example by Jonathan Murphy and Charles Gran 2007
sr      = 44100
ksmps   = 10
```

```

nchnls      = 2

; new, and important. Make sure that midi note events are only
; received by instruments that actually need them.

; turn default midi routing off
massign      0, 0
; route note events on channel 1 to instr 1
massign      1, 1

; Define your midi controllers
#define C1 #21#
#define C2 #22#
#define C3 #23#

; Initialize MIDI controllers
initc7      1, $C1, 0.5           ;delay send
initc7      1, $C2, 0.5           ;delay: time to zero
initc7      1, $C3, 0.5           ;delay: rate

gaosc      init      0

; Define an opcode to "smooth" the MIDI controller signal
opcode      smooth, k, k
kin         xin
kport       linseg      0, 0.0001, 0.01, 1, 0.01
kin         portk      kin, kport
            xout      kin
endop

instr      1
; Generate a sine wave at the frequency of the MIDI note that triggered the instrument
ifgc      cpsmidi
iamp      ampmidi      10000
aenv      linenr      iamp, .01, .1, .01           ;envelope
al        oscil      aenv, ifgc, 1
; All sound goes to the global variable gaosc
gaosc      = gaosc + al
endin

instr      198 ; ECHO
kcmbsnd      ctrl7      1, $C1, 0, 1           ;delay send
ktime      ctrl7      1, $C2, 0.01, 6           ;time loop fades out
kloop      ctrl7      1, $C3, 0.01, 1           ;loop speed
; Receive MIDI controller values and then smooth them
kcmbsnd      smooth      kcmbsnd
ktime      smooth      ktime
kloop      smooth      kloop
imaxlpt      = 1           ;max loop time
; Create a variable reverberation (delay) of the gaosc signal
acomb      vcomb      gaosc, ktime, kloop, imaxlpt, 1
aout      = (acomb * kcmbsnd) + gaosc * (1 - kcmbsnd)
            outs      aout, aout
gaosc      = 0
endin

</CsInstruments>
<CsScore>
f1 0 16384 10 1
i198 0 10000
e
</CsScore>
</CsoundSynthesizer>

```

## See Also

*alpass, comb, reverb, valpass*

## Credits

Author: William « Pete » Moss  
University of Texas at Austin

Austin, Texas USA  
January 2002

# vcopy

vcopy — Copies between two vectorial control signals

## Description

Copies between two vectorial control signals

## Syntax

```
vcopy ifn, ifn2, kelements [, kdstoffset] [, ksrcoffset] [, kverbose]
```

## Initialization

*ifn1* - number of the table where the vectorial signal will be copied (destination)

*ifn2* - number of the table hosting the vectorial signal to be copied (source)

## Performance

*kelements* - number of elements of the vector

*kdstoffset* - index offset for the destination (*ifn1*) table (Default=0)

*ksrcoffset* - index offset for the source (*ifn2*) table (Default=0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vcopy* copies *kelements* elements from *ifn2* (starting from position *ksrcoffset*) to *ifn1* (starting from position *kdstoffset*). Useful to keep old vector values, by storing them in another table.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *kdstoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are copied). There's an i-rate version of this opcode called *vcopy\_i*.



### Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate.

This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

All these operators (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexp*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

*Note:* *bmscan* not yet available on Canonical Csound

## Examples

Here is an example of the *vcopy* opcode. It uses the file *vcopy.csd* [examples/vcopy.csd].

### Exemple 558. Example of the *vcopy* opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vcopy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2

instr 1 ;table playback
ar lposcil 1, 1, 0, 262144, 1
outs ar,ar
endin

instr 2
vcopy 2, 1, 20000 ;copy vector from sample to empty table
vmult 5, 20000, 262144 ;scale noise to make it audible
vcopy 1, 5, 20000 ;put noise into sample
turnoff
endin

instr 3
vcopy 1, 2, 20000 ;put original information back in
turnoff
endin

</CsInstruments>
<CsScore>
f1 0 262144 -1 "beats.wav" 0 4 0
f2 0 262144 2 0

f5 0 262144 21 3 30000

i1 0 4
i2 3 1

s
i1 0 4
i3 3 1
```

```
s  
i1 0 4  
</CsScore>  
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)



# vcopy\_i

`vcopy_i` — Copies a vector from one table to another.

## Description

Copies a vector from one table to another.

## Syntax

```
vcopy_i ifn, ifn2, ielements [,idstoffset, isrcoffset]
```

## Initialization

*ifn* - number of the table where the vectorial signal will be copied

*ifn* - number of the table hosting the vectorial signal to be copied

*ielements* - number of elements of the vector

*idstoffset* - index offset for destination table

*isrcoffset* - index offset for source table

## Performance

`vcopy` copies *ielements* elements from *ifn2* (starting from position *isrcoffset*) to *ifn1* (starting from position *idstoffset*). Useful to keep old vector values, by storing them in another table. This opcode is exactly the same as `vcopy` but performs all the copying on the initialization pass only.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will be set to 0). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector elements will be 0).



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

All these operators (`vaddv`, `vsubv`, `vmultv`, `vdivv`, `vpowv`, `vexp`, `vcopy` and `vmap`) are designed to be used together with other opcodes that operate with vectorial signals such as `bmscan`, `vcella`, `adsynt`, `adsynt2` etc.

*Note:* `bmscan` not yet available on Canonical Csound

## Examples

See `vcopy` for an example.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vdelay

vdelay — Un délai variable avec interpolation.

## Description

C'est un délai variable avec interpolation qui n'est pas très différent de l'implémentation existante (*delta-pi*), il est simplement plus facile à utiliser.

## Syntaxe

```
ares vdelay asig, adel, imaxdel [, iskip]
```

## Initialisation

*imaxdel* -- Valeur maximale du délai en millisecondes. Si *adel* reçoit une valeur supérieure à *imaxdel* celle-ci est repliée autour de *imaxdel*. Cela est à éviter.

*iskip* -- L'initialisation est ignorée s'il est présent et différent de zéro.

## Exécution

Avec ce générateur unitaire il est possible de faire des effets Doppler ou de chorus et de flanger.

*asig* -- Signal en entrée.

*adel* -- Valeur courante du délai en millisecondes. Noter que les fonctions linéaires n'ont pas d'effet de modification de la hauteur. Des valeurs de *adel* changeant rapidement provoqueront des discontinuités dans la forme d'onde ce qui donne du bruit.

## Exemples

```
f1 0 8192 10 1
ims      =      100          ; Maximum delay time in msec
a1       oscil      10000, 1737, 1 ; Make a signal
a2       oscil      ims/2, 1/p3, 1 ; Make an LFO
a2       =      a2 + ims/2      ; Offset the LFO so that it is positive
a3       vdelay    a1, a2, ims    ; Use the LFO to control delay time
out
```

Deux points importants ici. D'abord, la valeur du retard doit toujours être positive. Ensuite, même si la valeur du retard peut être contrôlée au taux-k, il n'est pas prudent d'agir ainsi, car des changements de durée soudains provoqueront des clics.

## Voir Aussi

*vdelay3*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# vdelay3

vdelay3 — Un délai variable avec interpolation cubique.

## Description

*vdelay3* est expérimental. Il est semblable à *vdelay* sauf qu'il utilise l'interpolation cubique. (Nouveau dans la version 3.50.)

## Syntaxe

```
ares vdelay3 asig, adel, imaxdel [, iskip]
```

## Initialisation

*imaxdel* -- Valeur maximale du délai en millisecondes. Si *adel* reçoit une valeur supérieure à *imaxdel* celle-ci est repliée autour de *imaxdel*. Cela est à éviter.

*iskip* (facultatif) -- L'initialisation est ignorée s'il est présent et différent de zéro.

## Exécution

Avec ce générateur unitaire il est possible de faire des effets Doppler ou de chorus et de flanger.

*asig* -- Signal en entrée.

*adel* -- Valeur courante du délai en millisecondes. Noter que les fonctions linéaires n'ont pas d'effet de modification de la hauteur. Des valeurs de *adel* changeant rapidement provoqueront des discontinuités dans la forme d'onde ce qui donne du bruit.

## Exemples

```
f1 0 8192 10 1
ims      =      100          ; Maximum delay time in msec
a1       oscil      10000, 1737, 1 ; Make a signal
a2       oscil      ims/2, 1/p3, 1 ; Make an LFO
a2       =      a2 + ims/2      ; Offset the LFO so that it is positive
a3       vdelay    a1, a2, ims    ; Use the LFO to control delay time
out
```

Deux points importants ici. D'abord, la valeur du retard doit toujours être positive. Ensuite, même si la valeur du retard peut être contrôlée au taux-k, il n'est pas prudent d'agir ainsi, car des changements de durée soudains provoqueront des clics.

## Voir Aussi

*vdelay*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# vdelayx

vdelayx — Un opcode de délai variable avec interpolation de grande qualité.

## Description

Un opcode de délai variable avec interpolation de grande qualité.

## Syntaxe

```
aout vdelayx ain, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*aout* -- signal audio en sortie.

*ain* -- signal audio en entrée.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.



### Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw\**, le changement de la durée du délai a des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$
  
où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.
- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Voir Aussi

*vdelayxq, vdelayxs, vdelayxw, vdelayxwq, vdelayxws*



# vdelayxq

vdelayxq — Un opcode de délai variable sur 4 canaux avec interpolation de grande qualité.

## Description

Un opcode de délai variable sur 4 canaux avec interpolation de grande qualité.

## Syntaxe

```
aout1, aout2, aout3, aout4 vdelayxq ain1, ain2, ain3, ain4, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*aout1, aout2, aout3, aout4* -- signaux audio en sortie.

*ain1, ain2, ain3, ain4* -- signaux audio en entrée.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes multicanaux (par exemple *vdelayxq*) permettent de retarder 2 ou 4 variables à la fois (signaux stéréo ou quadro) ; c'est bien plus efficace que d'utiliser un opcode séparé pour chaque canal.



## Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw\**, le changement de la durée du délai à des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$
  
où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.

- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Voir Aussi

*vdelayx, vdelayxs, vdelayxw, vdelayxwq, vdelayxws*

# vdelayxs

vdelayxs — Un opcode de délai variable stéréo avec interpolation de grande qualité.

## Description

Un opcode de délai variable stéréo avec interpolation de grande qualité.

## Syntaxe

```
aout1, aout2 vdelayxs ain1, ain2, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*aout1*, *aout2* -- signaux audio en sortie.

*ain1*, *ain2* -- signaux audio en entrée.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes multicanaux (par exemple *vdelayxq*) permettent de retarder 2 ou 4 variables à la fois (signaux stéréo ou quadro) ; c'est bien plus efficace que d'utiliser un opcode séparé pour chaque canal.



## Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw\**, le changement de la durée du délai à des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$
  
où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.

- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Voir Aussi

*vdelayx, vdelayxq, vdelayxw, vdelayxwq, vdelayxws*

# vdelayxw

vdelayxw — Opcode de délai variable avec interpolation de grande qualité.

## Description

Opcode de délai variable avec interpolation de grande qualité.

## Syntaxe

```
aout vdelayxw ain, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*aout* -- signal audio en sortie.

*ain* -- signal audio en entrée.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes *vdelayxw*\* changent la position d'écriture dans la ligne à retard (au contraire de tous les autres générateurs unitaires de délai qui déplacent la position de lecture), et sont particulièrement utiles pour implémenter l'effet Doppler dans lequel la position de l'auditeur est fixe alors que la source est en mouvement.



### Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de  $iws/2$  échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw*\*, le changement de la durée du délai à des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$

où  $a$  est le gain en sortie et  $dt$  est la valeur du changement du délai par seconde.

- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Voir Aussi

*vdelayx, vdelayxq, vdelayxs, vdelayxwq, vdelayxws*

# vdelayxwq

vdelayxwq — Opcode de délai variable avec interpolation de grande qualité.

## Description

Opcode de délai variable avec interpolation de grande qualité.

## Syntaxe

```
aout1, aout2, aout3, aout4 vdelayxwq ain1, ain2, ain3, ain4, adl, \  
imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*ain1, ain2, ain3, ain4* -- signaux audio en entrée.

*aout1, aout2, aout3, aout4* -- signaux audio en sortie.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes *vdelayxw\** changent la position d'écriture dans la ligne à retard (au contraire de tous les autres générateurs unitaires de délai qui déplacent la position de lecture), et sont particulièrement utiles pour implémenter l'effet Doppler dans lequel la position de l'auditeur est fixe alors que la source est en mouvement.

Les opcodes multicanaux (par exemple *vdelayxq*) permettent de retarder 2 ou 4 variables à la fois (signaux stéréo ou quadro) ; c'est bien plus efficace que d'utiliser un opcode séparé pour chaque canal.



## Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws/2* échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.

- Dans *vdelayxw\**, le changement de la durée du délai à des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$
  
où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.
- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Voir Aussi

*vdelayx, vdelayxq, vdelayxs, vdelayxw, vdelayxws*



# vdelayxws

vdelayxws — Opcode de délai variable avec interpolation de grande qualité.

## Description

Opcode de délai variable avec interpolation de grande qualité.

## Syntaxe

```
aout1, aout2 vdelayxws ain1, ain2, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* -- (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*ain1, ain2* -- signaux audio en entrée.

*aout1, aout2* -- signaux audio en sortie.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes *vdelayxw\** changent la position d'écriture dans la ligne à retard (au contraire de tous les autres générateurs unitaires de délai qui déplacent la position de lecture), et sont particulièrement utiles pour implémenter l'effet Doppler dans lequel la position de l'auditeur est fixe alors que la source est en mouvement.

Les opcodes multicanaux (par exemple *vdelayxq*) permettent de retarder 2 ou 4 variables à la fois (signaux stéréo ou quadro) ; c'est bien plus efficace que d'utiliser un opcode séparé pour chaque canal.



## Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.

- Dans *vdelayxw\**, le changement de la durée du délai à des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$
  
où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.
- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Voir Aussi

*vdelayx, vdelayxq, vdelayxs, vdelayxw, vdelayxwq*

# vdivv

vdivv — Performs division between two vectorial control signals

## Description

Performs division between two vectorial control signals

## Syntax

```
vdivv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

## Performance

*kelements* - number of elements of the two vectors

*kdstoffset* - index offset for the destination (ifn1) table (Default=0)

*ksrcoffset* - index offset for the source (ifn2) table (Default=0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vdivv* divides two vectorial control signals, that is, each element of *ifn1* is divided by the corresponding element of *ifn2*. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will be set to 0). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination elements will be set to 0).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are divided). There's an i-rate ver-

sion of this opcode called *vdivv\_i*.



## Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

All these operators (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Examples

Here is an example of the *vdivv* opcode. It uses the file *vdivv.csd* [examples/vdivv.csd].

### Exemple 559. Example of the *vdivv* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vdivv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
```

```
    endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16
f 2 0 16 -7 1 15 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vdivv\_i

vdivv\_i — Performs division between two vectorial control signals at init time.

## Description

Performs division between two vectorial control signals at init time.

## Syntax

```
vdivv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

*ielements* - number of elements of the two vectors

*idstoffset* - index offset for the destination (*ifn1*) table (Default=0)

*isrcoffset* - index offset for the source (*ifn2*) table (Default=0)

## Performance

*vdivv\_i* divides two vectorial control signals, that is, each element of *ifn1* is divided by the corresponding element of *ifn2*. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vdivv*.

All these operators (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vdelayk

vdelayk — k-rate variable time delay.

## Description

Variable delay applied to a k-rate signal

## Syntax

```
kout vdelayk  ksig, kdel, imaxdel [, iskip, imode]
```

## Initialization

*imaxdel* - maximum value of delay in seconds.

*iskip* (optional) - Skip initialization if present and non zero.

*imode* (optional) - if non-zero it suppresses linear interpolation. While, normally, interpolation increases the quality of a signal, it should be suppressed if using *vdelay* with discrete control signals, such as, for example, trigger signals.

## Performance

*kout* - delayed output signal

*ksig* - input signal

*kdel* - delay time in seconds can be varied at k-rate

*vdelayk* is similar to *vdelay*, but works at k-rate. It is designed to delay control signals, to be used, for example, in algorithmic composition.

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)



# vecdelay

vecdelay — Vectorial Control-rate Delay Paths

## Description

Generate a sort of 'vectorial' delay

## Syntax

```
vecdelay ifn, ifnIn, ifnDel, ielements, imaxdel [, iskip]
```

## Initialization

*ifn* - number of the table containing the output vector

*ifnIn* - number of the table containing the input vector

*ifnDel* - number of the table containing a vector whose elements contain delay values in seconds

*ielements* - number of elements of the two vectors

*imaxdel* - Maximum value of delay in seconds.

*iskip* (optional) - initial disposition of delay-loop data space (see *reson*). The default value is 0.

## Performance

*vecdelay* is similar to *vdelay*, but it works at k-rate and, instead of delaying a single signal, it delays a vector. *ifnIn* is the input vector of signals, *ifn* is the output vector of signals, and *ifnDel* is a vector containing delay times for each element, expressed in seconds. Elements of *ifnDel* can be updated at k-rate. Each single delay can be different from that of the other elements, and can vary at k-rate. *imaxdel* sets the maximum delay allowed for all elements of *ifnDel*.

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# veloc

veloc — Donne la vélocité d'un évènement MIDI.

## Description

Donne la vélocité d'un évènement MIDI.

## Syntaxe

```
ival veloc [ilow] [, ihigh]
```

## Initialisation

*ilow, ihigh* -- Limites basse et haute pour le mappage

## Exécution

Donne la valeur de l'octet MIDI (0 - 127) pour la vélocité de l'évènement courant.

## Exemples

Voici un exemple de l'opcode veloc. Il utilise le fichier *veloc.csd* [examples/veloc.csd].

### Exemple 560. Exemple le l'opcode veloc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages  MIDI in
-odac        -iadc       -d           -M0    ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o veloc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  il veloc

  print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 12 seconds.
i 1 0 12
```

e

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par Kevin Conder.

# vexp

vexp — Performs power-of operations between a vector and a scalar

## Description

Performs power-of operations between a vector and a scalar

## Syntax

```
vexp ifn, kval, kelements [, kdstoffset] [, kverbose]
```

## Initialization

*ifn* - number of the table hosting the vectorial signal to be processed

## Performance

*kval* - scalar operand to be processed

*kelements* - number of elements of the vector

*kdstoffset* - index offset for the destination table (Optional, default = 0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vexp* rises *kval* to each element contained in a vector from table *ifn*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Note that this opcode runs at k-rate so the value of *kval* is processed every control period. Use with care or you will end up with very large (or small) numbers (or use *vexp\_i*).

These opcodes (*vadd*, *vmult*, *vpow* and *vexp*) perform numeric operations between a vectorial control signal (hosted by the table *ifn*), and a scalar signal (*kval*). Result is a new vector that overrides old values of *ifn*. All these opcodes work at k-rate.

Negative values for *kdstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the initial vectors. If you want to keep initial vector, use *vcopy* or *vcopy\_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.



### Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *iele-*

ments is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

## Examples

Here is an example of the vexp opcode. It uses the file *vexp.csd* [examples/vexp.csd].

### Exemple 561. Example of the vexp opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vexp ifn1, ival, ielements, idstoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
```

e

```
</CsScore>  
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vexp\_i

vexp\_i — Performs power-of operations between a vector and a scalar

## Description

Performs power-of operations between a vector and a scalar

## Syntax

```
vexp_i ifn, ival, ielements[, idstoffset]
```

## Initialization

*ifn* - number of the table hosting the vectorial signal to be processed

*ielements* - number of elements of the vector

*ival* - scalar value to be added

*idstoffset* - index offset for the destination table

## Performance

*vexp\_i* rises *kval* to each element contained in a vector from table *ifn*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Negative values for *idstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

This opcode runs only on initialization, there is a k-rate version of this opcode called *vexp*.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the initial vectors. If you want to keep initial vector, use *vcopy* or *vcopy\_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

## Examples

Here is an example of the *vexp\_i* opcode. It uses the file *vexp\_i.csd* [examples/vexp\_i.csd].

### Exemple 562. Example of the vexp\_i opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac        -iadc        ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

    instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vexp_i ifn1, ival, ielements, idstoffset
    endin

    instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
    turnoff
endif

kcount = kcount + 1
    endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

## See also

*vadd*, *vmult\_i*, *vpow\_i* and *vexp\_i*.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)



# vexpseg

vexpseg — Vectorial envelope generator

## Description

Generate exponential vectorial segments

## Syntax

```
vexpseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]
```

## Initialization

*ifnout* - number of table hosting output vectorial signal

*ifn1* - starting vector

*ifn2, ifn3, etc.* - vector after *idurx* seconds

*idur1* - duration in seconds of first segment.

*dur2, idur3, etc.* - duration in seconds of subsequent segments.

*ielements* - number of elements of vectors.

## Performance

These opcodes are similar to *linseg* and *expseg*, but operate with vectorial signals instead of with scalar signals.

Output is a vectorial control signal hosted by *ifnout* (that must be previously allocated), while each break-point of the envelope is actually a vector of values. All break-points must contain the same number of elements (*ielements*).

All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Example

Here is an example of the *vexpseg* opcode. It uses the files *vexpseg.csd* [examples/vexpseg.csd].

### Exemple 563. Example of the vexpseg opcode.

```
<CsoundSynthesizer>  
<CsOptions>  
-odac -B441 -b441  
</CsOptions>  
<CsInstruments>
```

```
sr=44100
```

```
ksmps=10
nchnls=2

gilen init 32

gitable1 ftgen 0, 0, gilen, 10, 1
gitable2 ftgen 0, 0, gilen, 10, 1

gitable3 ftgen 0, 0, gilen, -7, 30, gilen, 35
gitable4 ftgen 0, 0, gilen, -7, 400, gilen, 450
gitable5 ftgen 0, 0, gilen, -7, 5000, gilen, 5500

instr 1
vcopy gitable2, gitable1, gilen
turnoff
endin

instr 2
vexpseg gitable2, 16, gitable3, 2, gitable4, 2, gitable5
endin

instr 3
kcount init 0
if kcount < 16 then
    kval table kcount, gitable2
    printk 0,kval
    kcount = kcount +1
else
    turnoff
endif
endin

</CsInstruments>
<CsScore>
i1 0 1
s
i2 0 10
i3 0 1
i3 1 1
i3 1.5 1
i3 2 1
i3 2.5 1
i3 3 1
i3 3.5 1
i3 4 1
i3 4.5 1

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado.

Example by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# vexpv

vexpv — Performs exponential operations between two vectorial control signals

## Description

Performs exponential operations between two vectorial control signals

## Syntax

```
vexpv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

## Performance

*kelements* - number of elements of the two vectors

*kdstoffset* - index offset for the destination (ifn1) table (Default=0)

*ksrcoffset* - index offset for the source (ifn2) table (Default=0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vexpv* elevates each element of *ifn2* to the corresponding element of *ifn1*. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will be set to 1). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination elements will be set to 1).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are processed). There's an i-rate version of this opcode called *vexpv\_i*.



## Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

All these operators (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Examples

Here is an example of the *vexpv* opcode. It uses the file *vexpv.csd* [examples/vexpv.csd].

### Exemple 564. Example of the *vexpv* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vexpv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
```

```
<CsScore>

f 1 0 16 -7 1 16 17
f 2 0 16 -7 0 16 1

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.002 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vexpv\_i

vexpv\_i — Performs exponential operations between two vectorial control signals at init time.

## Description

Performs exponential operations between two vectorial control signals at init time.

## Syntax

```
vexpv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

*ielements* - number of elements of the two vectors

*idstoffset* - index offset for the destination (ifn1) table (Default=0)

*isrcoffset* - index offset for the source (ifn2) table (Default=0)

## Performance

*vexpv\_i* elevates each element of ifn2 to the corresponding element of ifn1. Each vectorial signal is hosted by a table (ifn1 and ifn2). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of ifn1. If you want to keep the old ifn1 vector, use *vcopy\_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will be set to 1). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector elements will be set to 1).



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vexpv*.

All these operators (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vibes

vibes — Modèle physique de la frappe d'un bloc de métal.

## Description

La sortie audio est un son de métal frappé comme sur un vibraphone. La méthode est un modèle physique développé d'après Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares vibes kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec
```

## Initialisation

*ihrd* -- la dureté de la baguette utilisé pour frapper. Compris entre 0 et 1. 0,5 est une valeur adaptée.

*ipos* -- l'endroit où le bloc est frappé, compris entre 0 et 1.

*imp* -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

*ivfn* -- forme du vibrato, habituellement une table sinus, créée par une fonction

*idec* -- durée avant la fin de la note lorsqu'il y a une atténuation

*idoubles* (facultatif) -- pourcentage de frappes doubles. La valeur par défaut est de 40%.

*itriples* (facultatif) -- pourcentage de frappes triples. La valeur par défaut est de 20%.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kvibf* -- Fréquence du vibrato en Hertz. L'intervalle conseillé va de 0 à 12.

*kvamp* -- Amplitude du vibrato.

## Exemples

Voici un exemple de l'opcode vibes. Il utilise les fichiers *vibes.csd* [examples/vibes.csd] et *marmstk1.wav* [examples/marmstk1.wav].

### Exemple 565. Exemple de l'opcode vibes.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vibes.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; kamp = 20000
; kfreq = 440
; ihrd = 0.5
; ipos = 0.561
; imp = 1
; kvibf = 6.0
; kvamp = 0.05
; ivibfn = 2
; idec = 0.1

a1 vibes 20000, 440, 0.5, 0.561, 1, 6.0, 0.05, 2, 0.1

out a1
endin

</CsInstruments>
<CsScore>

; Table #1, the "marmstkl.wav" audio file.
f 1 0 256 1 "marmstkl.wav" 0 0 0
; Table #2, a sine wave for the vibrato.
f 2 0 128 10 1

; Play Instrument #1 for four seconds.
i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*marimba*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# vibr

vibr — Vibrato contrôlable par l'utilisateur, d'usage plus facile.

## Description

Vibrato contrôlable par l'utilisateur, d'usage plus facile.

## Syntaxe

```
kout vibr kAverageAmp, kAverageFreq, ifn
```

## Initialisation

*ifn* -- Numéro de la table de vibrato. Elle contient normalement une onde sinus ou triangle.

## Exécution

*kAverageAmp* -- Valeur d'amplitude moyenne du vibrato

*kAverageFreq* -- Valeur de fréquence moyenne du vibrato (en cps)

*vibr* est une version de *vibrato* d'usage plus facile. Il a le même moteur de génération que *vibrato*, mais les paramètres correspondant aux arguments d'entrée manquants sont codés en dur sur des valeurs par défaut.

## Exemples

Voici un exemple de l'opcode vibr. Il utilise le fichier *vibr.csd* [examples/vibr.csd].

### Exemple 566. Exemple de l'opcode vibr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vibr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create a vibrato waveform.
kaverageamp init 7500
kaveragefreq init 5
```

```
ifn = 1
kvamp vibr kaverageamp, kaveragefreq, ifn

; Generate a tone including the vibrato.
a1 oscili 10000+kvamp, 440, 2

out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vibrato.
f 1 0 256 10 1
; Table #1, a sine wave for the oscillator.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*jitter, jitter2, vibrato*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15

# vibrato

vibrato — Génère un vibrato naturel contrôlable par l'utilisateur.

## Description

Génère un vibrato naturel contrôlable par l'utilisateur.

## Syntaxe

```
kout vibrato kAverageAmp, kAverageFreq, kRandAmountAmp, \  
      kRandAmountFreq, kAmpMinRate, kAmpMaxRate, kcpsMinRate, \  
      kcpsMaxRate, ifn [, iphs]
```

## Initialisation

*ifn* -- Numéro de la table de vibrato. Elle contient normalement une onde sinus ou triangle.

*iphs* -- (facultatif) Phase initiale de la table, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*kAverageAmp* -- Valeur de l'amplitude moyenne du vibrato

*kAverageFreq* -- Valeur de la fréquence moyenne du vibrato (en cps)

*kRandAmountAmp* -- Importance de la déviation aléatoire de l'amplitude

*kRandAmountFreq* -- Importance de la déviation aléatoire de la fréquence

*kAmpMinRate* -- Fréquence minimale des segments de déviation aléatoire de l'amplitude (en cps)

*kAmpMaxRate* -- Fréquence maximale des segments de déviation aléatoire de l'amplitude (en cps)

*kcpsMinRate* -- Fréquence minimale des segments de déviation aléatoire de la fréquence (en cps)

*kcpsMaxRate* -- Fréquence maximale des segments de déviation aléatoire de la fréquence (en cps)

*vibrato* produit un vibrato naturel contrôlable par l'utilisateur. Le concept consiste à varier aléatoirement la fréquence et l'amplitude de l'oscillateur générant le vibrato, afin de simuler les irrégularités d'un vibrato réel.

Afin d'avoir un contrôle total de ces variations aléatoires, plusieurs arguments sont présents en entrée. Les variations aléatoires sont obtenues à partir de deux suites séparées de segments, la première contrôlant les déviations d'amplitude, la seconde les déviations de fréquence. La durée moyenne de chaque segment dans chaque suite peut être raccourcie ou allongée par les arguments *kAmpMinRate*, *kAmpMaxRate*, *kcpsMinRate*, *kcpsMaxRate*, et les déviations par rapport aux valeurs d'amplitude et de fréquence moyennes peuvent être ajustées indépendamment au moyen de *kRandAmountAmp* et de *kRandAmountFreq*.

## Exemples

Voici un exemple de l'opcode vibrato. Il utilise le fichier *vibrato.csd* [exemples/vibrato.csd].

### Exemple 567. Example of the vibrato opcode.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vibrato.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create a vibrato waveform.
kaverageamp init 2500
kaveragefreq init 6
krandamountamp init 0.3
krandamountfreq init 0.5
kampminrate init 3
kampmaxrate init 5
kcpsminrate init 3
kcpsmaxrate init 5
ifn = 1
kvamp vibrato kaverageamp, kaveragefreq, krandamountamp, \
             krandamountfreq, kampminrate, kampmaxrate, \
             kcpsminrate, kcpsmaxrate, ifn

; Generate a tone including the vibrato.
a1 oscili 10000+kvamp, 440, 2

out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave for the vibrato.
f 1 0 256 10 1
; Table #1, a sine wave for the oscillator.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*jitter, jitter2, vibr*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15

# vincr

vincr — Accumule des signaux audio.

## Description

*vincr* incrémente une variable audio avec un autre signal, c-à-d qu'il accumule les valeurs dans sa sortie.

## Syntaxe

```
vincr accum, aincr
```

## Exécution

*accum* -- variable accumulateur de taux-a à incrémenter

*aincr* -- signal d'incrémentation

*vincr* (variable increment) et *clear* sont prévus pour être utilisés ensemble. *vincr* stocke la somme de deux variables audio dans la première variable (qui joue ainsi le rôle d'un accumulateur en polyphonie). L'accumulateur est habituellement une variable globale qui est utilisée pour combiner des signaux provenant de plusieurs sources (différents instruments ou instances d'instruments) pour un traitement ultérieur (par exemple via un effet global qui lit l'accumulateur) ou pour sortir le signal composé par un autre moyen que les opcodes *out* (par exemple via l'opcode *fout*). Après son utilisation, la variable accumulateur doit être remise à zéro au moyen de l'opcode *clear* (sinon elle sera saturée).

## Exemples

Voir l'exemple de l'opcode *fout*.

## Voir Aussi

*clear*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# vlimit

vlimit — Limiting and Wrapping Vectorial Signals

## Description

Limits elements of vectorial control signals.

## Syntax

```
vlimit ifn, kmin, kmax, ielements
```

## Initialization

*ifn* - number of the table hosting the vector to be processed

*ielements* - number of elements of the vector

## Performance

*kmin* - minimum threshold value

*kmax* - maximum threshold value

*vlimit* set lower and upper limits on each element of the vector they process.

These opcodes are similar to *limit*, *wrap* and *mirror*, but operate with a vectorial signal instead of with a scalar signal.

Result overrides old values of *ifn1*, if these are out of min/max interval. If you want to keep input vector, use *vcopy* opcode to copy it in another table.

All these opcodes are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

*Note:* *bmscan* not yet available on Canonical Csound

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)



# vlinseg

vlinseg — Vectorial envelope generator

## Description

Generate linear vectorial segments

## Syntax

```
vlinseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]
```

## Initialization

*ifnout* - number of table hosting output vectorial signal

*ifn1* - starting vector

*ifn2, ifn3, etc.* - vector after *idurx* seconds

*idur1* - duration in seconds of first segment.

*dur2, idur3, etc.* - duration in seconds of subsequent segments.

*ielements* - number of elements of vectors.

## Performance

These opcodes are similar to `linseg` and `expseg`, but operate with vectorial signals instead of with scalar signals.

Output is a vectorial control signal hosted by *ifnout* (that must be previously allocated), while each break-point of the envelope is actually a vector of values. All break-points must contain the same number of elements (*ielements*).

All these operators are designed to be used together with other opcodes that operate with vectorial signals such as `bmscan`, `vcella`, `adsynt`, `adsynt2` etc.

## Example

Here is an example of the `vlinseg` opcode. It uses the files `vlinseg.csd` [examples/vlinseg.csd].

### Exemple 568. Example of the vlinseg opcode.

```
<CsoundSynthesizer>  
<CsOptions>  
-odac -B441 -b441  
</CsOptions>  
<CsInstruments>
```

```
sr=44100
```

```
ksmps=10
nchnls=2

gilen init 32

gitable1 ftgen 0, 0, gilen, 10, 1
gitable2 ftgen 0, 0, gilen, 10, 1

gitable3 ftgen 0, 0, gilen, -7, 30, gilen, 35
gitable4 ftgen 0, 0, gilen, -7, 400, gilen, 450
gitable5 ftgen 0, 0, gilen, -7, 5000, gilen, 5500

instr 1
vcopy gitable2, gitable1, gilen
turnoff
endin

instr 2
vlinseg gitable2, 16, gitable3, 2, gitable4, 2, gitable5
endin

instr 3
kcount init 0
if kcount < 16 then
    kval table kcount, gitable2
    printk 0,kval
    kcount = kcount +1
else
    turnoff
endif
endin

</CsInstruments>
<CsScore>
i1 0 1
s
i2 0 10
i3 0 1
i3 1 1
i3 1.5 1
i3 2 1
i3 2.5 1
i3 3 1
i3 3.5 1
i3 4 1
i3 4.5 1

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado.

Example by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# vlowres

vlowres — A bank of filters in which the cutoff frequency can be separated under user control.

## Description

A bank of filters in which the cutoff frequency can be separated under user control

## Syntax

```
ares vlowres asig, kfco, kres, iord, ksep
```

## Initialization

*iord* -- total number of filters (1 to 10)

## Performance

*asig* -- input signal

*kfco* -- frequency cutoff (not in Hz)

*ksep* -- frequency cutoff separation for each filter

*vlowres* (variable resonant lowpass filter) allows a variable response curve in resonant filters. It can be thought of as a bank of lowpass resonant filters, each with the same resonance, serially connected. The frequency cutoff of each filter can vary with the *kfco* and *ksep* parameters.

## Examples

Here is an example of the vlowres opcode. It uses the file *vlowres.csd* [examples/vlowres.csd], and *beats.wav* [examples/beats.wav].

### Exemple 569. Example of the vlowres opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vlowres.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
; Use a nice sawtooth waveform.
asig vco 32000, 220, 1

; Vary the cutoff frequency from 30 to 300 Hz.
kfco line 30, p3, 300
kres = 25
iord = 2
ksep = 20

; Apply the filters.
avlr vlowres asig, kfco, kres, iord, ksep

; It gets loud, so clip the output amplitude to 30,000.
al clip avlr, 1, 30000
out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Author: Gabriel Maldonado  
Italy

Example written by Kevin Conder.

New in Csound version 3.49

# vmap

vmap — Maps elements from a vector according to indices contained in another vector

## Description

Maps elements from a vector onto another according to the indices of a this vector

## Syntax

```
vmap ifn1, ifn2, ielements [,idstoffset, isrcoffset]
```

## Initialization

*ifn1* - number of the table where the vectorial signal will be copied, and which contains the mapping vector

*ifn2* - number of the table hosting the vectorial signal to be copied

*ielements* - number of elements to process

*idstoffset* - index offset for destination table (*ifn1*)

*isrcoffset* - index offset for source table (*ifn2*)

## Performance

*vmap* maps elements of *ifn2* according to the values of table *ifn1*. Elements of *ifn1* are treated as indexes of table *ifn2*, so element values of *ifn1* must not exceed the length of *ifn2* table otherwise a Csound will report an error. Elements of *ifn1* are treated as integers, so any fractional part will be truncated. There is no interpolation performed on this operation.

In practice, what happens is that the elements of *ifn1* are used as indices to *ifn2*, and then are replaced by the corresponding elements from *ifn2*. *ifn1* must be different from *ifn2*, otherwise the results are unpredictable. Csound will produce an init error if they are not.

All these operators (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

*Note:* *bmscan* not yet available on Canonical Csound

## Examples

Here is an example of the *vmap* opcode. It uses the file *vmap.csd* [examples/vmap.csd].

### Exemple 570. Example of the *vmap* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vmap.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
ksmps = 256
nchnls = 2
gisize = 64

gitable ftgen 0, 0, gisize, 10, 1 ;Table to be processed
gimap1 ftgen 0, 0, gisize, -7, gisize-1, gisize-1, 0 ; Mapping function to reverse table
gimap2 ftgen 0, 0, gisize, -5, 1, gisize-1, gisize-1 ; Mapping function for PWM
gimap3 ftgen 0, 0, gisize, -7, 1, (gisize/2)-1, gisize-1, 1, 1, (gisize/2)-1, gisize-1 ; Double frequency

instr 1 ;Hear an oscillator using gitable
asig oscil 10000, 440, gitable
outs asig,asig
endin

instr 2 ;Reverse the table (no sound change, except for a single click
vmap gimap1, gitable, gisize
vcopy_i gitable, gimap1, gisize
turnoff
endin

instr 3 ;Non-interpolated PWM (or phase waveshaping)
vmap gimap2, gitable, gisize
vcopy_i gitable, gimap2, gisize
turnoff
endin

instr 4 ;Double frequency
vmap gimap3, gitable, gisize
vcopy_i gitable, gimap3, gisize
turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 8

i 2 2 1
i 3 4 1
i 4 6 1

e
</CsScore>
</CsoundSynthesizer>

```

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vmirror

vmirror — Limiting and Wrapping Vectorial Signals

## Description

'Reflects' elements of vectorial control signals on thresholds.

## Syntax

```
vmirror ifn, kmin, kmax, ielements
```

## Initialization

*ifn* - number of the table hosting the vector to be processed

*ielements* - number of elements of the vector

## Performance

*kmin* - minimum threshold value

*kmax* - maximum threshold value

*vmirror* 'reflects' each element of corresponding vector if it exceeds low or high thresholds.

These opcodes are similar to *limit*, *wrap* and *mirror*, but operate with a vectorial signal instead of with a scalar signal.

Result overrides old values of *ifn1*, if these are out of min/max interval. If you want to keep input vector, use *vcopy* opcode to copy it in another table.

All these opcodes are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

*Note:* *bmscan* not yet available on Canonical Csound

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vmult

vmult — Multiplies a vector in a table by a scalar value.

## Description

Multiplies a vector in a table by a scalar value.

## Syntax

```
vmult ifn, kval, kelements [, kdstoffset] [, kverbose]
```

## Initialization

*ifn* - number of the table hosting the vectorial signal to be processed

## Performance

*kval* - scalar value to be multiplied

*kelements* - number of elements of the vector

*kdstoffset* - index offset for the destination table (Optional, default = 0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vmult* multiplies each element of the vector contained in the table *ifn* by *kval*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Note that this opcode runs at k-rate so the value of *kval* is multiplied every control period. Use with care or you will end up with very large numbers (or use *vmult\_i*).

These opcodes (*vadd*, *vmult*, *vpow* and *vexp*) perform numeric operations between a vectorial control signal (hosted by the table *ifn*), and a scalar signal (*kval*). Result is a new vector that overrides old values of *ifn*. All these opcodes work at k-rate.

Negative values for *kdstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the initial vectors. If you want to keep initial vector, use *vcopy* or *vcopy\_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.



### Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *iele-*



ments is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

## See also

*vadd\_i*, *vadd*, *vmult\_i*, *vpow* and *vexp*.

## Example

Here is an example of the *vmult* opcode. It uses the file *vmult-2.csd* [examples/vmult-2.csd].

### Exemple 571. Example of the *vmult* opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vmult ifn1, ival, ielements, idstoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
```

```
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Here is another example of the vmult opcode. It uses the file *vmult.csd* [examples/vmult.csd].

### Exemple 572. Example of the vmult opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2

        instr 1 ;table playback
ar lposcil 1, 1, 0, 262144, 1
outs ar,ar
        endin

        instr 2
vcopy 2, 1, 40000 ;copy vector from sample to empty table
vmult 5, 10000, 262144 ;scale noise to make it audible
vcopy 1, 5, 40000 ;put noise into sample
turnoff
        endin

        instr 3
vcopy 1, 2, 40000 ;put original information back in
turnoff
        endin

</CsInstruments>
<CsScore>
f1 0 262144 -1 "beats.wav" 0 4 0
f2 0 262144 2 0

f5 0 262144 21 3 30000

i1 0 4
i2 3 1

s
i1 0 4
i3 3 1
s

i1 0 4

</CsScore>
</CsoundSynthesizer>
```

## See also

*vadd\_i*, *vmult*, *vpow* and *vexp*.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

Example by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# vmult\_i

vmult\_i — Multiplies a vector in a table by a scalar value.

## Description

Multiplies a vector in a table by a scalar value.

## Syntax

```
vmult_i ifn, ival, ielements [, idstoffset]
```

## Initialization

*ifn* - number of the table hosting the vectorial signal to be processed

*ival* - scalar value to be multiplied

*ielements* - number of elements of the vector

*idstoffset* - index offset for the destination table

## Performance

*vmult\_i* multiplies each element of the vector contained in the table *ifn* by *ival*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

This opcode runs only on initialization, there is a k-rate version of this opcode called *vmult*.

Negative values for *idstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the initial vectors. If you want to keep initial vector, use *vcopy* or *vcopy\_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

## Examples

Here is an example of the *vmult\_i* opcode. It uses the file *vmult\_i.csd* [examples/vmult\_i.csd].

### Exemple 573. Example of the vmult\_i opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in
-odac        -iadc        ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

    instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vmult_i ifn1, ival, ielements, idstoffset
    endin

    instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
    turnoff
endif

kcount = kcount + 1
    endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>

```

## See also

*vadd*, *vadd*, *vmult*, *vpow* and *vexp*.

## See also

*vadd\_i*, *vmult*, *vpow\_i* and *vexp\_i*.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

Example by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# vmultv

vmultv — Performs multiplication between two vectorial control signals

## Description

Performs multiplication between two vectorial control signals

## Syntax

```
vmultv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

## Performance

*kelements* - number of elements of the two vectors

*kdstoffset* - index offset for the destination (*ifn1*) table (Default=0)

*ksrcoffset* - index offset for the source (*ifn2*) table (Default=0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vmultv* multiplies two vectorial control signals, that is, each element of the first vector is processed (only) with the corresponding element of the other vector. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The Result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are multiplied). There's an i-rate

version of this opcode called *vmultv\_i*.



## Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

All these operators (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Examples

Here is an example of the *vmultv* opcode. It uses the file *vmultv.csd* [examples/vmultv.csd].

### Exemple 574. Example of the *vmultv* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vmultv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin
```

```
</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17
f 2 0 16 -7 1 16 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)



# vmultv\_i

vmultv\_i — Performs multiplication between two vectorial control signals at init time.

## Description

Performs multiplication between two vectorial control signals at init time.

## Syntax

```
vmultv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

*ielements* - number of elements of the two vectors

*idstoffset* - index offset for the destination (*ifn1*) table (Default=0)

*isrcoffset* - index offset for the source (*ifn2*) table (Default=0)

## Performance

*vmultv\_i* multiplies two vectorial control signals, that is, each element of the first vector is processed (only) with the corresponding element of the other vector. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vmultv*.

All these operators (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# voice

voice — Simulation d'une voix humaine.

## Description

Simulation d'une voix humaine.

## Syntaxe

ares **voice** kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn

## Initialisation

*ifn*, *ivfn* -- numéros des deux tables contenant la forme d'onde de la porteuse et la forme d'onde du vibrato. Les fichiers *impuls20.aiff* [examples/impuls20.aiff], *ahh.aiff* [examples/ahh.aiff], *eee.aiff* [examples/eee.aiff] ou *ooo.aiff* [examples/ooo.aiff] conviennent pour la première, et la deuxième peut contenir une sinusoïde. Ces fichiers sont disponibles à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Frequency de la note. Elle peut varier pendant l'exécution.

*kphoneme* -- un entier compris entre 0 et 16, pour choisir les formants des sons :

- « eee », « ihh », « ehk », « aaa »,
- « ahh », « aww », « ohh », « uhh »,
- « uuu », « ooo », « rrr », « lll »,
- « mmm », « nnn », « nng », « ngg ».

Actuellement les phonèmes

- « fff », « sss », « thh », « shh »,
- « xxx », « hee », « hoo », « hah »,
- « bbb », « ddd », « jjj », « ggg »,
- « vvv », « zzz », « thz », « zhh »

ne sont pas disponibles (!)

*kform* -- gain pour le phonème. Des valeurs entre 0,0 et 1,2 sont recommandées.

*kvibf* -- fréquence du vibrato en Hertz. On suggère des valeurs entre 0 et 12

*kvamp* -- amplitude du vibrato

## Exemples

Voici un exemple de l'opcode *voice*. Il utilise les fichiers *voice.csd* [examples/voice.csd] et *impuls20.aiff* [examples/impuls20.aiff].

### Exemple 575. Exemple de l'opcode *voice*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o voice.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 22050
kr = 2205
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 3
  kfreq = 0.8
  kphoneme = 6
  kform = 0.488
  kvibf = 0.04
  kvamp = 1
  ifn = 1
  ivfn = 2

  av voice kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn

; It tends to get loud, so clip voice's amplitude at 30,000.
al clip av, 2, 30000
out al
endin

</CsInstruments>
<CsScore>

; Table #1, an audio file for the carrier waveform.
f 1 0 256 1 "impuls20.aiff" 0 0 0
; Table #2, a sine wave for the vibrato waveform.
f 2 0 256 10 1

; Play Instrument #1 for a half-second.
i 1 0 0.5
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitich (d'après Perry Cook)

Université de Bath, Codemist Ltd.  
Bath, UK

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47 de Csound

# vosim

vosim — Simulation vocale simple basée sur des pulsations glottales avec des caractéristiques de formant.

## Description

Cet opcode produit une simulation vocale simple basée sur des pulsations glottales avec des caractéristiques de formant. La sortie est une suite d'évènements sonores dans laquelle chaque élément est composé d'une explosion de pulsations sinusoïdales élevées au carré suivies par un silence. La méthode de synthèse VOSIM (VOcal SIMulation) fut développée par Kaegi et Tempelaars dans les années 1970.

## Syntaxe

ar **vosim** kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]

## Intialisation

*ifn* - une table sonore contenant normalement une demie-période d'une onde sinusoïdale, élevée au carré (voir les notes ci-dessous).

*iskip* - (facultatif) L'initialisation est ignorée, pour les notes liées.

## Exécution

*ar* - signal en sortie. Noter que la sortie est habituellement unipolaire - seulement positive.

*kamp* - amplitude de la sortie, l'amplitude de crête de la première pulsation dans chaque explosion.

*kFund* - hauteur fondamentale, en Hz. Chaque événement dure  $1/kFund$  secondes.

*kForm* - fréquence du formant central. La longueur de chaque pulsation dans l'explosion vaut  $1/kForm$  secondes.

*kDecay* - facteur d'amortissement d'une pulsation à l'autre. Il est soustrait de l'amplitude à chaque nouvelle pulsation.

*kPulseCount* - nombre de pulsations dans la partie explosive de chaque événement.

*kPulseFactor* - la largeur de pulsation est multipliée par cette valeur à chaque nouvelle pulsation. Cela provoque un glissement de formant. Si le factor est  $< 1.0$ , le formant monte, s'il est  $> 1.0$  chaque nouvelle pulsation est plus longue et ainsi le format descend. La hauteur finale du formant vaut  $kForm * \text{pow}(kPulseFactor, kPulseCount)$

La sortie de *vosim* est une suite d'évènements sonores, dans laquelle chaque événement est composé d'une explosion de pulsations sinusoïdales élevées au carré suivies par un silence. La durée totale des événements détermine la fréquence fondamentale. La longueur de chaque impulsion individuelle dans l'explosion de sinus au carré produit une bande de fréquence formantique. La largeur du formant est déterminée par le taux de silence par rapport aux pulsations (voir ci-dessous). Le résultat final est aussi modelé par le facteur d'atténuation entre pulsations.

Le fait qu'aucune fonction GEN ne crée une onde sinusoïdale élevée au carré telle quelle pose un petit problème dans l'utilisation de cet opcode. On peut créer la table appropriée depuis la partition en utili-

sant quelque chose comme ce qui suit.

```
; use GEN09 to create half a sine in table 17
f 17 time size 9 0.5 1 0
; run instr 101 on table 17 for a single init-pass
i 101 0 0 17
```

On peut aussi le faire avec un instrument qui remplit une f-table dans l'orchestre :

```
; square each point in table #p4. This should be run as init-only, just once in the performance
instr 101
  index tableng p4
  index = index - 1 ; start from last point
loop:
  ival table index, p4
  ival = ival * ival
  tableiw ival, index, p4
  index = index - 1
  if index < 0 igoto endloop
                    igoto loop
endloop:
endin
```



## Limites de Paramètre

Le nombre de pulsations multiplié par la largeur de pulsation doit être inclus dans la longueur de l'évènement ( $1/kFund$ ). Si ce n'est pas le cas, l'algorithme fonctionne quand même, mais les pulsations qui se trouveraient en dehors de l'évènement ne sont pas démarrées. Cela peut introduire un silence à la fin de l'évènement même s'il n'est pas désiré. En conséquence,  $kForm$  doit être supérieur à  $kFund$ , sinon il n'y aura que du silence en sortie.

*Vosim* a été créé pour émuler des sons vocaux en modélisant des impulsions glottales. On peut créer des sons riches en combinant plusieurs instances de *vosim* avec différents paramètres. Le fait que le signal ne soit pas à bande limitée est un inconvénient. Mais comme les auteurs le souligne, l'atténuation des composants aigus est de -60 dB à six fois la fréquence fondamentale. On peut également modifier le signal en changeant le signal source dans la table de lecture. La technique a un intérêt historique et peut produire des sons riches à moindre frais (chaque échantillon ne nécessite qu'une lecture dans la table suivie d'une seule multiplication pour l'atténuation).

Comme indiqué, la largeur de bande du formant dépend du rapport entre l'explosion de pulsation et le silence dans un évènement. Mais ce n'est pas un paramètre indépendant : la fondamentale fixe la longueur de l'évènement tandis que le centre du formant définit la longueur de la pulsation. Il est ainsi impossible de garantir un rapport explosion/silence spécifique, car la longueur de l'explosion doit être un multiple entier de la longueur de la pulsation. La chute des pulsations peut être utilisée pour lisser la transition de  $N$  à  $N \pm 1$  pulsations, mais il y aura toujours des paliers dans le profil spectral de la sortie. L'exemple de code ci-dessous montre une telle approche.

Tous les paramètres en entrée sont de taux-k. Les paramètres en entrée ne sont utilisés que pour déterminer chaque nouvel évènement (ou grain). L'amplitude de l'évènement est fixée pour chaque évènement à l'initialisation. Pour les valeurs usuelles des paramètres, lorsque  $ksmps < 500$ , les paramètres de taux-k sont mis à jour plus souvent que les évènements ne sont créés. Dans tous les cas, il n'y aura pas de bruit à large bande injecté dans le système à cause d'entrées de taux-k mises à jour moins souvent qu'elles ne sont lues, mais quelques artefacts peuvent être créés.

L'opcode devrait se comporter raisonnablement pour toutes les entrées. Quelques détails :

- a.  $kFund < 0$  : il est forcé à une valeur positive - pas de points dans des événements "inversés".
- b.  $kFund == 0$  : cela conduit à un événement de longueur "infinie", c'est-à-dire une explosion de pulsation suivie par un très long silence indéfini.
- c.  $kForm == 0$  : cela conduit à une pulsation de longueur infinie, ainsi aucune pulsation n'est générée (c'est-à-dire silence).
- d.  $kForm < 0$  : la table est lue à l'envers. Si la table est symétrique,  $kform$  et  $-kform$  donneront des sorties identiques bit à bit.
- e.  $kPulseFactor == 0$  : la seconde pulsation en avant est zéro. Voir (c).
- f.  $kPulseFactor < 0$  : les pulsations lisent la table alternativement à l'endroit et à l'envers.

Avec une table de pulsation asymétrique, un  $kForm$  ou un  $kPulseFactor$  négatifs peuvent être utiles.

## Exemples

Voici un exemple de l'opcode vosim. Il utilise le fichier *vosim.csd* [examples/vosim.csd].

### Exemple 576. Exemple de l'opcode vosim.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o vosim.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps   = 100
nchnls  = 1

#####
; By Rasmus Ekman 2008

; Square each point in table #p4. This should only be run once in the performance.
instr 10

    index tableng p4
    index = index - 1 ; start from last point
loop:
    ival table index, p4
    ival = ival * ival
    tableiw ival, index, p4
    index = index - 1
    if index < 0 igoto endloop
                igoto loop
endloop:
endin

#####
; Main vosim instrument. Sweeps from a fund1/form1 to fund2/form2,
; trying for narrowest formant bandwidth (still quite wide by the looks of it)
; p4:      amp
```



```

; p5, p6: fund beg-end
; p7, p8: form beg-end
; p9: amp decay (ignored)
; p10: pulse count (ignored - calc internally)
; p11: pulse length mod
; p12: skip (for tied events)
; p13: don't fade out (if followed by tied note)
instr 1
  kamp init p4
  ; freq start, end
  kfund line p5, p3, p6
  ; formant start, end
  kform line p7, p3, p8

  ; Try for constant ratio burst/silence, and narrowest formant bandwidth
  kPulseCount = (kform / kfund) ;init p10
  ; Attempt to smooth steps between format bandwidths,
  ; increasing decay before we are forced to a lower pulse count
  kDecay = kPulseCount/(kform % kfund) ; init p9
  if (kDecay * kPulseCount) > kamp then
    kDecay = kamp / kPulseCount
  endif
  kDecay = 0.3 * kDecay

  kPulseFactor init p11

; ar vosim kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]
  arl vosim kamp, kfund, kform, kDecay, kPulseCount, kPulseFactor, 17, p12

  ; scale amplitude for 16-bit files, with quick fade out
  amp init 20000
  if (p13 != 0) goto nofade
  amp linseg 20000, p3-.02, 20000, .02, 0
nofade:
  out arl * amp
endin

</CsInstruments>
<CsScore>

f1 0 32768 9 1 1 0 ; sine wave
f17 0 32768 9 0.5 1 0 ; half sine wave
i10 0 0 17 ; init run only, square table 17

; Vosim score

; Picking some formants from the table in Csound manual

; p4=amp fund form decay pulses pulsemod [skip] nofade
; tenor a -> e
i1 0 .5 .5 280 240 650 400 .03 5 1
i1 . . .3 . . 1080 1700 .03 5 .
i1 . . .2 . . 2650 2600 .03 5 .
i1 . . .15 . . 2900 3200 .03 5 .

; tenor a -> o
i1 0.6 .2 .5 300 210 650 400 .03 5 1 0 1
i1 . . .3 . . 1080 800 .03 5 . . .
i1 . . .2 . . 2650 2600 .03 5 . . .
i1 . . .15 . . 2900 2800 .03 5 . . .

; tenor o -> aah
i1 .8 .3 .5 210 180 400 650 .03 5 1 1 1
i1 . . .3 . . 800 1080 .03 5 . . .
i1 . . .2 . . 2600 2650 .03 5 . . .
i1 . . .15 . . 2800 2900 .03 5 . . .

; tenor aa -> i
i1 1.1 .2 .5 180 250 650 290 .03 5 1 1 1
i1 . . .3 . . 1080 1870 .03 5 . . .
i1 . . .2 . . 2650 2800 .03 5 . . .
i1 . . .15 . . 2900 3250 .03 5 . . .

; tenor i -> u
i1 1.3 .3 .5 250 270 290 350 .03 5 1 1 0
i1 . . .3 . . 1870 600 .03 5 . . .
i1 . . .2 . . 2800 2700 .03 5 . . .
i1 . . .15 . . 3250 2900 .03 5 . . .

e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*fof, fof2*

## Crédits

Auteur : Rasmus Ekman  
Mars 2008

# vphaseseg

vphaseseg — Allows one-dimensional HVS (Hyper-Vectorial Synthesis).

## Description

*vphaseseg* allows one-dimensional HVS (Hyper-Vectorial Synthesis).

## Syntax

```
vphaseseg kphase, ioutab, ielems, itab1, idist1, itab2 \
[, idist2, itab3, ... , idistN-1, itabN]
```

## Initialization

*ioutab* - number of output table.

*ielem* - number of elements to process

*itab1*, ..., *itabN* - breakpoint table numbers

*idist1*, ..., *idistN-1* - distances between breakpoints in percentage values

## Performance

*kphase* - phase pointer

*vphaseseg* returns the coordinates of section points of an N-dimensional space path. The coordinates of section points are stored into an output table. The number of dimensions of the N-dimensional space is determined by the *ielem* argument that is equal to N and can be set to any number. To define the path, user have to provide a set of points of the N-dimensional space, called break-points. Coordinates of each break-point must be contained by a different table. The number of coordinates to insert in each break-point table must obviously equal to *ielem* argument. There can be any number of break-point tables filled by the user.

Hyper-Vectorial Synthesis actually deals with two kinds of spaces. The first space is the N-dimensional space in which the path is defined, this space is called time-variant parameter space (or SPACE A). The path belonging to this space is covered by moving a point into the second space that normally has a number of dimensions smaller than the first. Actually, the point in motion is the projection of corresponding point of the N-dimensional space (could also be considered a section of the path). The second space is called user-pointer-motion space (or SPACE B) and, in the case of *vphaseseg* opcode, has only ONE DIMENSION. Space B is covered by means of *kphase* argument (that is a sort of path pointer), and its range is 0 to 1. The output corresponding to current pointer value is stored in *ioutab* table, whose data can be afterwards used to control any synthesis parameters.

In *vphaseseg*, each break-point is separated from the other by a distance expressed in percentage, where all the path length is equal to the sum of all distances. So distances between breakpoints can be different, differently from kinds of HVS in which space B has more than one dimension, in these cases distance between break-points MUST be THE SAME for all intervals.

## See Also

*hvs1, hvs2, hvs3*

## Credits

Author: Gabriel Maldonado

New in version 5.06

# vport

vport — Vectorial Control-rate Delay Paths

## Description

Generate a sort of 'vectorial' portamento

## Syntax

```
vport ifn, khtime, ielements [, ifnInit]
```

## Initialization

*ifn* - number of the table containing the output vector

*ielements* - number of elements of the two vectors

*ifnInit* (optional) - number of the table containing a vector whose elements contain initial portamento values.

## Performance

*vport* is similar to *port*, but operates with vectorial signals, instead of with scalar signals. Each vector element is treated as an independent control signal. Input vector input and output vectors are placed in the same table and output vector overrides input vector. If you want to keep input vector, use *vcopy* opcode to copy it in another table.

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vpow

vpow — Raises each element of a vector to a scalar power

## Description

Raises each element of a vector to a scalar power

## Syntax

```
vpow ifn, kval, kelements [, kdstoffset] [, kverbose]
```

## Initialization

*ifn* - number of the table hosting the vectorial signal to be processed

## Performance

*kval* - scalar value to which the elements of *ifn* will be raised

*kelements* - number of elements of the vector

*kdstoffset* - index offset for the destination table (Optional, default = 0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vpow* raises each element of the vector contained in the table *ifn* to the power of *kval*, starting from table index *kdstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

Note that this opcode runs at k-rate so the value of *kval* is processed every control period. Use with care or you will end up with very large (or small) numbers (or use *vpow\_i*).

These opcodes (*vadd*, *vmult*, *vpow* and *vexp*) perform numeric operations between a vectorial control signal (hosted by the table *ifn*), and a scalar signal (*kval*). Result is a new vector that overrides old values of *ifn*. All these opcodes work at k-rate.

Negative values for *kdstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the initial vectors. If you want to keep initial vector, use *vcopy* or *vcopy\_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.



### Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *iele-*

ments is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

## Examples

Here is an example of the vpow opcode. It uses the file *vpow.csd* [examples/vpow.csd].

### Exemple 577. Example of the vpow opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vpow ifn1, ival, ielements, idstoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## See also

*vadd\_i*, *vmult*, *vpow* and *vexp*.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)



# vpow\_i

vpow\_i — Raises each element of a vector to a scalar power

## Description

Raises each element of a vector to a scalar power

## Syntax

```
vpow_i ifn, ival, ielements [, idstoffset]
```

## Initialization

*ifn* - number of the table hosting the vectorial signal to be processed

*ielements* - number of elements of the vector

*ival* - scalar value to which the elements of *ifn* will be raised

*idstoffset* - index offset for the destination table

## Performance

*vpow\_i* elevates each element of the vector contained in the table *ifn* to the power of *ival*, starting from table index *idstoffset*. This enables you to process a specific section of a table by specifying the offset and the number of elements to be processed. Offset is counted starting from 0, so if no offset is specified (or set to 0), the table will be modified from the beginning.

This opcode runs only on initialization, there is a k-rate version of this opcode called *vpow*.

Negative values for *idstoffset* are valid. Elements from the vector that are outside the table, will be discarded, and they will not wrap around the table.

In all these opcodes, the resulting vectors are stored in *ifn*, overriding the initial vectors. If you want to keep initial vector, use *vcopy* or *vcopy\_i* to copy it in another table. All these operators are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2*, etc. They can also be useful in conjunction with the spectral opcodes *pvsftw* and *pvsftr*.

## Examples

Here is an example of the *vpow\_i* opcode. It uses the file *vpow\_i.csd* [examples/vpow\_i.csd].

### Exemple 578. Example of the vpow\_i opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in
-odac        -iadc        ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

        instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vpow_i ifn1, ival, ielements, idstoffset
        endin

        instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
    turnoff
endif

kcount = kcount + 1
        endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>
</CsoundSynthesizer>

```

## See also

*vadd\_i*, *vmult\_i*, *vpow* and *vexp\_i*.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vpowv

vpowv — Performs power-of operations between two vectorial control signals

## Description

Performs power-of operations between two vectorial control signals

## Syntax

```
vpowv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

## Performance

*kelements* - number of elements of the two vectors

*kdstoffset* - index offset for the destination (*ifn1*) table (Default=0)

*ksrcoffset* - index offset for the source (*ifn2*) table (Default=0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vpowv* raises each element of *ifn1* to the corresponding element of *ifn2*. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are processed). There's an i-rate version of this opcode called *vpowv\_i*.



## Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *elements* is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

All these operators (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Examples

Here is an example of the *vpowv* opcode. It uses the file *vpowv.csd* [examples/vpowv.csd].

### Exemple 579. Example of the *vpowv* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vpowv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
```

```
<CsScore>

f 1 0 16 -7 1 16 17

f 2 0 16 -7 1 16 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vpowv\_i

`vpowv_i` — Performs power-of operations between two vectorial control signals at init time.

## Description

Performs power-of operations between two vectorial control signals at init time.

## Syntax

```
vpowv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

*ielements* - number of elements of the two vectors

*idstoffset* - index offset for the destination (*ifn1*) table

*isrcoffset* - index offset for the source (*ifn2*) table

## Performance

`vpowv_i` raises each element of *ifn1* to the corresponding element of *ifn2*. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use `vcopy_i` opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 1 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 1 (i.e. the destination vector will not be changed for these elements).



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called `vpowv`.

All these operators (`vaddv_i`, `vsubv_i`, `vmultv_i`, `vdivv_i`, `vpowv_i`, `vexpv_i`, `vcopy` and `vmap`) are designed to be used together with other opcodes that operate with vectorial signals such as `bmscan`, `vcella`, `adsynt`, `adsynt2` etc.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

## vpvoc

vpvoc — Implements signal reconstruction using an fft-based phase vocoder and an extra envelope.

## Description

Implements signal reconstruction using an fft-based phase vocoder and an extra envelope.

## Syntax

```
ares vpvoc ktmpnt, kfmod, ifile [, ispecwp] [, ifn]
```

## Initialization

*ifile* -- the pvoc number (n in pvoc.n) or the name in quotes of the analysis file made using pvanal. (See *pvoc*.)

*ispecwp* (optional, default=0) -- if non-zero, attempts to preserve the spectral envelope while its frequency content is varied by *kfmod*. The default value is zero.

*ifn* (optional, default=0) -- optional function table containing control information for vpvoc. If *ifn* = 0, control is derived internally from a previous *tableseg* or *tablexseg* unit. Default is 0. (New in Csound version 3.59)

## Performance

*ktmpnt* -- The passage of time, in seconds, through the analysis file. *ktmpnt* must always be positive, but can move forwards or backwards in time, be stationary or discontinuous, as a pointer into the analysis file.

*kfmod* -- a control-rate transposition factor: a value of 1 incurs no transposition, 1.5 transposes up a perfect fifth, and .5 down an octave.

This implementation of *pvoc* was originally written by Dan Ellis. It is based in part on the system of Mark Dolson, but the pre-analysis concept is new. The spectral extraction and amplitude gating (new in Csound version 3.56) were added by Richard Karpen based on functions in SoundHack by Tom Erbe.

*vpvoc* is identical to *pvoc* except that it takes the result of a previous *tableseg* or *tablexseg* and uses the resulting function table (passed internally to the *vpvoc*), as an envelope over the magnitudes of the analysis data channels. Optionally, a table specified by *ifn* may be used.

The result is spectral enveloping. The function size used in the *tableseg* should be *framesize/2*, where *framesize* is the number of bins in the phase vocoder analysis file that is being used by the *vpvoc*. Each location in the table will be used to scale a single analysis bin. By using different functions for *ifn1*, *ifn2*, etc.. in the *tableseg*, the spectral envelope becomes a dynamically changing one. See also *tableseg* and *tablexseg*.

## Examples

The following example, using *vpvoc*, shows the use of functions such as



```
f 1 0 256 5 .001 128 1 128 .001
f 2 0 256 5 1 128 .001 128 1
f 3 0 256 7 1 256 1
```

to scale the amplitudes of the separate analysis bins.

```
ptime    line          0, p3,3 ; time pointer, in seconds, into file
tablexseg 1, p3*.5, 2, p3*.5, 3
apv       vpvoc        ktime,1, "pvoc.file"
```

The result would be a time-varying « spectral envelope » applied to the phase vocoder analysis data. Since this amplifies or attenuates the amount of signal at the frequencies that are paired with the amplitudes which are scaled by these functions, it has the effect of applying very accurate filters to the signal. In this example the first table would have the effect of a band-pass filter, gradually be band-rejected over half the note's duration, and then go towards no modification of the magnitudes over the second half.

## See Also

*pvoc*

## Credits

Authors: Dan Ellis and Richard Karpen  
Seattle, WA USA  
1997

New in version 3.44

# vrandh

**vrandh** — Generates a vector of random numbers stored into a table, holding the values for a period of time.

## Description

Generates a vector of random numbers stored into a table, holding the values for a period of time. Generates a sort of 'vectorial band-limited noise'.

## Syntax

```
vrandh ifn, krange, kcps, ielements [, idstoffset] [, iseed]  
          [, isize] [, ioffset]
```

## Initialization

*ifn* - number of the table where the vectorial signal will be generated

*ielements* - number of elements of the vector

*idstoffset* - (optional, default=0) -- index offset for the destination table

*iseed* (optional, default=0.5) -- seed value for the recursive pseudo-random formula. A value between 0 and +1 will produce an initial output of  $kamp * iseed$ . A negative value will cause seed re-initialization to be skipped. A value greater than 1 will seed from system time, this is the best option to generate a different random sequence for each run.

*isize* (optional, default=0) -- if zero, a 16 bit number is generated. If non-zero, a 31-bit random number is generated. Default is 0.

*ioffset* - (optional, default=0) -- a base value added to the random result.

## Performance

*krange* - range of random elements (from -krange to krange)

*kcps* - rate of generated elements in cycles per seconds

This opcode is similar to *randh*, but operates on vectors instead of with scalar values.

Though the argument *isize* defaults to 0, thus using a 16-bit random number generator, using the newer 31-bit algorithm is recommended, as this will produce a random sequence with a longer period (more random numbers before the sequence starts repeating).

The output is a vector contained in *ifn* (that must be previously allocated).

All these operators are designed to be used together with other opocdes that operate with vector such as *bmscan*, *adsynt* etc.

*Note:* *bmscan* not yet available on Canonical Csound

## Examples

Here is an example of the `vrandh` opcode. It uses the file `vrandh.csd` [examples/vrandh.csd].

### Exemple 580. Example of the `vrandh` opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vrandh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Andres Cabrera

sr=44100
ksmps=128
nchnls=2

gitab ftgen 0, 0, 16, -7, 0, 128, 0

instr 1
  krange init p4
  kcps init p5
  ioffset init p6

  kav1 init 0
  kav2 init 0
  kcount init 0

  ;      table      krange kcps  ielements  idstoffset  iseed  isize ioffset
  vrandh gitab, krange, kcps,      3,          3,          2,  0,  ioffset

  kfreq1 table 3, gitab
  kfreq2 table 4, gitab
  kfreq3 table 5, gitab

  ;Change the frequency of three oscillators according to the random values
  aosc1 oscili 4000, kfreq1, 1
  aosc2 oscili 2000, kfreq2, 1
  aosc3 oscili 4000, kfreq3, 1

  outs aosc1+aosc2, aosc3+aosc2
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
;
i 1 0          5 100 1 300
i 1 5          5 300 1 400
i 1 10         5 100 2 1000
i 1 15         5 400 4 1000
i 1 20         5 1000 8 2000
i 1 25         5 250 16 300
e

</CsScore>
</CsSoundSynthesizer>
```

## See also

*vrandi, randh*

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vrandi

vrandi — Generate a sort of 'vectorial band-limited noise'

## Description

Generate a sort of 'vectorial band-limited noise'

## Syntax

```
vrandi ifn, krange, kcps, ielements [, idstoffset] [, iseed]  
        [, isize] [, ioffset]
```

## Initialization

*ifn* - number of the table where the vectorial signal will be generated

*ielements* - number of elements to process

*idstoffset* - (optional, default=0) -- index offset for the destination table

*iseed* (optional, default=0.5) -- seed value for the recursive pseudo-random formula. A value between 0 and +1 will produce an initial output of  $kamp * iseed$ . A negative value will cause seed re-initialization to be skipped. A value greater than 1 will seed from system time, this is the best option to generate a different random sequence for each run.

*isize* (optional, default=0) -- if zero, a 16 bit number is generated. If non-zero, a 31-bit random number is generated. Default is 0.

*ioffset* - (optional, default=0) -- a base value added to the random result.

## Performance

*krange* - range of random elements (from -krange to krange)

*kcps* - rate of generated elements in cycles per seconds

This opcode is similar to *randi*, but operates on vectors instead of with scalar values.

Though argument *isize* defaults to 0, thus using a 16-bit random number generator, using the newer 31-bit algorithm is recommended, as this will produce a random sequence with a longer period (more random numbers before the sequence starts repeating).

The output is a vector contained in *ifn* (that must be previously allocated).

All these operators are designed to be used together with other opocdes that operate with vector such as *bmscan*, *adsynt* etc.

*Note:* *bmscan* not yet available on Canonical Csound

## Examples

Here is an example of the *vrandi* opcode. It uses the file *vrandi.csd* [examples/vrandi.csd].

## Exemple 581. Example of the vrandi opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vrandi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera

gitab ftgen 0, 0, 16, -7, 0, 128, 0

instr 1
  krange init p4
  kcps init p5
  ioffset init p6
  ;      table      krange kcps  ielements  idstoffset  iseed  isize ioffset
  vrandi gitab, krange, kcps,      3,          3,          2,  1,  ioffset

  kfreq1 table 3, gitab
  kfreq2 table 4, gitab
  kfreq3 table 5, gitab

  ;Change the frequency of three oscillators according to the random values
  aosc1 oscili 4000, kfreq1, 1
  aosc2 oscili 2000, kfreq2, 1
  aosc3 oscili 4000, kfreq3, 1

  outs aosc1+aosc2, aosc3+aosc2
endin

</CsInstruments>
<CsScore>

f 1 0 2048 10 1

;
i 1 0      krange kcps  ioffset
5 100 1 300
i 1 5      5 5 1 400
i 1 10     5 100 2 1000
i 1 15     5 400 4 1000
i 1 20     5 1000 8 2000
i 1 20     5 300 32 350

e

</CsScore>
</CsoundSynthesizer>
```

## See also

*vrandh*, *randi*

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vstaudio, vstaudiog

vstaudio — VST audio output.

## Syntax

```
aout1,aout2 vstaudio instance, [ain1, ain2]
```

```
aout1,aout2 vstaudiog instance, [ain1, ain2]
```

## Description

*vstaudio* and *vstaudiog* are used for sending and receiving audio from a VST plugin.

*vstaudio* is used within an instrument definition that contains a *vstmidiout* or *vstnote* opcode. It outputs audio for only that one instrument. Any audio remaining in the plugin after the end of the note, for example a reverb tail, will be cut off and should be dealt with using a damping envelope.

*vstaudiog* (*vstaudio* global) is used in a separate instrument to process audio from any number of VST notes or MIDI events that share the same VST plugin instance (*instance*). The *vstaudiog* instrument must be numbered higher than all the instruments receiving notes or MIDI data, and the note controlling the *vstplug* instrument must have an indefinite duration, or at least a duration as long as the VST plugin is active.

## Initialization

*instance* - the number which identifies the plugin, to be passed to other *vst4cs* opcodes.

## Performance

*aout1*, *aout2* - the audio output received from the plugin.

*ain1*, *ain2* - the audio input sent to the plugin.

## Examples

Here is an example of the use of the *vstaudio* opcode. It uses the file *vst4cs.csd* [examples/vst4cs.csd].

### Exemple 582. Example of the *vstaudio* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Credits: Adapted by Michael Gogins
; from code by David Horowitz and Lian Cheung.
; The "--displays" option is required in order for
; the Pianoteq GUI to dispatch events and display properly.
-m3 --displays -odac
</CsOptions>
<CsInstruments>
```



```

sr      = 44100
ksmps   = 20
nchnls  = 2

; Load the Pianoteq into memory.
gipianoteq vstinit      "C:\\Program Files\\Steinberg\\VstPlugins\\Pianoteq 3.0 Trial\\Pianoteq

; Print information about the Pianoteq, such as parameter names and numbers.
vstinfo      gipianoteq

; Open the Pianoteq's GUI.
vstedit      gipianoteq

; Send notes from the score to the Pianoteq.
instr 1
; MIDI channels are numbered starting at 0.
; p3 always contains the duration of the note.
; p4 contains the MIDI key number (pitch),
; p5 contains the MIDI velocity number (loudness),
imidichannel init      0
vstnote      gipianoteq, imidichannel, p4, p5, p3
endin

; Send parameter changes to the Pianoteq.
instr 2
; p4 is the parameter number.
; p5 is the parameter value.
vstparamset  gipianoteq, p4, p5
endin

; Send audio from the Pianoteq to the output.
instr 3
init          0
ablankinput vstaudio  gipianoteq, ablankinput, ablankinput
aleft, aright outs    aleft, aright
endin

</CsInstruments>
<CsScore>
; Turn on the instrument that receives audio from the Pianoteq indefinitely.
i 3 0 -1
; Send parameter changes to Pianoteq before sending any notes.
; NOTE: All parameters must be between 0.0 and 1.0.
; Length of piano strings:
i 2 0 1 33 0.5
; Hammer noise:
i 2 0 1 25 0.1
; Send a C major 7th arpeggio to the Pianoteq.
i 1 1 10 60 76
i 1 2 10 64 73
i 1 3 10 67 70
i 1 4 10 71 67
; End the performance, leaving some time
; for the Pianoteq to finish sending out its audio,
; or for the user to play with the Pianoteq virtual keyboard.
e 20
</CsScore>
</CsoundSynthesizer>

```

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

# vstbankload

vstbankload — Loads parameter banks to a VST plugin.

## Syntax

```
vstbankload instance, ipath
```

## Description

*vstbankload* is used for loading parameter banks to a VST plugin.

## Initialization

*instance* - the number which identifies the plugin, to be passed to other vst4cs opcodes.

*ipath* - the full pathname of the parameter bank (.fxb file).

## Examples

### Exemple 583. Example for vstbankload

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
gihandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 4
vstbankload gihandle1,"c:/vstplugins/cheeze/chengo'scheese.fxb"
vstinfo gihandle1
endin

/* sco */
i 3 0 21
i4 1 1 57 32
```

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

# vstedit

vstedit — Opens the GUI editor widow for a VST plugin.

## Syntax

**vstedit** instance

## Description

*vstedit* opens the custom GUI editor widow for a VST plugin. Note that not all VST plugins have custom GUI editors. It may be necessary to use the `--displays` command-line option to ensure that Csound handles events from the editor window and displays it properly.

## Initialization

*instance* - the number which identifies the plugin, to be passed to other *vst4cs* opcodes.

## Examples

Here is an example of the use of the *vstedit* opcode. It uses the file *vst4cs.csd* [examples/vst4cs.csd].

### Exemple 584. Example of the *vstedit* opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Credits: Adapted by Michael Gogins
; from code by David Horowitz and Lian Cheung.
; The "--displays" option is required in order for
; the Pianoteq GUI to dispatch events and display properly.
-m3 --displays -odac
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps   = 20
nchnls  = 2

gipianoteq      ; Load the Pianoteq into memory.
vstinit        "C:\\Program Files\\Steinberg\\VstPlugins\\Pianoteq 3.0 Trial\\Pianoteq

; Print information about the Pianoteq, such as parameter names and numbers.
vstinfo        gipianoteq

; Open the Pianoteq's GUI.
vstedit        gipianoteq

; Send notes from the score to the Pianoteq.
instr 1
; MIDI channels are numbered starting at 0.
; p3 always contains the duration of the note.
; p4 contains the MIDI key number (pitch),
; p5 contains the MIDI velocity number (loudness),
imidichannel    init      0
vstnote         gipianoteq, imidichannel, p4, p5, p3
endin

; Send parameter changes to the Pianoteq.
```

```

instr 2
; p4 is the parameter number.
; p5 is the parameter value.
vstparamset      gipianoteq, p4, p5
endin

; Send audio from the Pianoteq to the output.
instr 3
init             0
vstaudio         gipianoteq, ablankinput, ablankinput
outs             aleft, aright
endin

ablaninput
aleft, aright

</CsInstruments>
<CsScore>
; Turn on the instrument that receives audio from the Pianoteq indefinitely.
i 3 0 -1
; Send parameter changes to Pianoteq before sending any notes.
; NOTE: All parameters must be between 0.0 and 1.0.
; Length of piano strings:
i 2 0 1 33 0.5
; Hammer noise:
i 2 0 1 25 0.1
; Send a C major 7th arpeggio to the Pianoteq.
i 1 1 10 60 76
i 1 2 10 64 73
i 1 3 10 67 70
i 1 4 10 71 67
; End the performance, leaving some time
; for the Pianoteq to finish sending out its audio,
; or for the user to play with the Pianoteq virtual keyboard.
e 20
</CsScore>
</CsoundSynthesizer>

```

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

# vstinit

vstinit — Load a VST plugin into memory for use with the other vst4cs opcodes.

## Syntax

```
instance vstinit ilibrarypath [,iverbose]
```

## Description

*vstinit* is used to load a VST plugin into memory for use with the other vst4cs opcodes. Both VST effects and instruments (synthesizers) can be used.

## Initialization

*instance* - the number which identifies the plugin, to be passed to other vst4cs opcodes.

*ilibrarypath* - the full path to the vst plugin shared library (DLL, on Windows). Remember to use '/' instead of '\' as separator.

*iverbose* - show plugin information and parameters when loading.

## Examples

Here is an example of the use of the *vstinit* opcode. It uses the file *vst4cs.csd* [examples/vst4cs.csd].

### Exemple 585. Example of the *vstinit* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsSoundSynthesizer>
<CsOptions>
; Credits: Adapted by Michael Gogins
; from code by David Horowitz and Lian Cheung.
; The "--displays" option is required in order for
; the Pianoteq GUI to dispatch events and display properly.
-m3 --displays -odac
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps   = 20
nchnls  = 2
; Load the Pianoteq into memory.
gipianoteq vstinit "C:\\Program Files\\Steinberg\\VstPlugins\\Pianoteq 3.0 Trial\\Pianoteq
; Print information about the Pianoteq, such as parameter names and numbers.
vstinfo    gipianoteq
; Open the Pianoteq's GUI.
vstedit    gipianoteq
; Send notes from the score to the Pianoteq.
instr 1
; MIDI channels are numbered starting at 0.
; p3 always contains the duration of the note.
; p4 contains the MIDI key number (pitch),
; p5 contains the MIDI velocity number (loudness),
```

```

imidichannel    init                0
                 vstnote            gipianoteq, imidichannel, p4, p5, p3
                 endin

                 ; Send parameter changes to the Pianoteq.
                 instr 2
                 ; p4 is the parameter number.
                 ; p5 is the parameter value.
                 vstparamset        gipianoteq, p4, p5
                 endin

                 ; Send audio from the Pianoteq to the output.
                 instr 3
ablankinput     init                0
aleft, aright   vstaudio            gipianoteq, ablankinput, ablankinput
                 outs                aleft, aright
                 endin

</CsInstruments>
<CsScore>
; Turn on the instrument that receives audio from the Pianoteq indefinitely.
i 3 0 -1
; Send parameter changes to Pianoteq before sending any notes.
; NOTE: All parameters must be between 0.0 and 1.0.
; Length of piano strings:
i 2 0 1 33 0.5
; Hammer noise:
i 2 0 1 25 0.1
; Send a C major 7th arpeggio to the Pianoteq.
i 1 1 10 60 76
i 1 2 10 64 73
i 1 3 10 67 70
i 1 4 10 71 67
; End the performance, leaving some time
; for the Pianoteq to finish sending out its audio,
; or for the user to play with the Pianoteq virtual keyboard.
e 20
</CsScore>
</CsoundSynthesizer>

```

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

# vstinfo

vstinfo — Displays the parameters and the programs of a VST plugin.

## Syntax

**vstinfo** instance

## Description

*vstinfo* displays the parameters and the programs of a VST plugin.

Note: The *verbose* flag in *vstinit* gives the same information as *vstinfo*. *vstinfo* is useful after loading parameter banks, or when the plugin changes parameters dynamically.

## Initialization

*instance* - the number which identifies the plugin, to be passed to other vst4cs opcodes.

## Examples

Here is an example of the use of the *vstinfo* opcode. It uses the file *vst4cs.csd* [examples/vst4cs.csd].

### Exemple 586. Example of the *vstinfo* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsSoundSynthesizer>
<CsOptions>
; Credits: Adapted by Michael Gogins
; from code by David Horowitz and Lian Cheung.
; The "--displays" option is required in order for
; the Pianoteq GUI to dispatch events and display properly.
-m3 --displays -odac
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps   = 20
nchnls  = 2

gipianoteq      ; Load the Pianoteq into memory.
vstinit         "C:\\Program Files\\Steinberg\\VstPlugins\\Pianoteq 3.0 Trial\\Pianoteq
                ; Print information about the Pianoteq, such as parameter names and numbers.
vstinfo         gipianoteq

                ; Open the Pianoteq's GUI.
vstedit         gipianoteq

                ; Send notes from the score to the Pianoteq.
instr 1
; MIDI channels are numbered starting at 0.
; p3 always contains the duration of the note.
; p4 contains the MIDI key number (pitch),
; p5 contains the MIDI velocity number (loudness),
imidichannel    init      0
vstnote         gipianoteq, imidichannel, p4, p5, p3
endin
```

```

; Send parameter changes to the Pianoteq.
instr 2
; p4 is the parameter number.
; p5 is the parameter value.
vstparamset      gipianoteq, p4, p5
endin

; Send audio from the Pianoteq to the output.
instr 3
ablankinput      init      0
aleft, aright    vstaudio   gipianoteq, ablankinput, ablankinput
outs             aleft, aright
endin

</CsInstruments>
<CsScore>
; Turn on the instrument that receives audio from the Pianoteq indefinitely.
i 3 0 -1
; Send parameter changes to Pianoteq before sending any notes.
; NOTE: All parameters must be between 0.0 and 1.0.
; Length of piano strings:
i 2 0 1 33 0.5
; Hammer noise:
i 2 0 1 25 0.1
; Send a C major 7th arpeggio to the Pianoteq.
i 1 1 10 60 76
i 1 2 10 64 73
i 1 3 10 67 70
i 1 4 10 71 67
; End the performance, leaving some time
; for the Pianoteq to finish sending out its audio,
; or for the user to play with the Pianoteq virtual keyboard.
e 20
</CsScore>
</CsoundSynthesizer>

```

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.



# vstmidiout

vstmidiout — Sends MIDI information to a VST plugin.

## Syntax

```
vstmidiout instance, kstatus, kchan, kdata1, kdata2
```

## Description

*vstmidiout* is used for sending MIDI information to a VST plugin.

## Initialization

*instance* - the number which identifies the plugin, to be passed to other vst4cs opcodes.

## Performance

*kstatus* - the type of midi message to be sent. Currently noteon (144), note off (128), Control Change (176), Program change (192), Aftertouch (208) and Pitch Bend (224) are supported.

*kchan* - the MIDI channel transmitted on.

*kdata1*, *kdata2* - the MIDI data pair, which varies depending on *kstatus*. e.g. note/velocity for note on and note off, Controller number/value for control change.

## Examples

### Exemple 587. Example for vstmidiout

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
gihandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 3
ain1 = 0
ab1, ab2 vstaudio gihandle1, ain1, ain1
outs ab1, ab2
endin
instr 4
vstmidiout gihandle1,144,1,p4,p5
endin
```

```
/* sco */
i 3 0 21
i4 1 1 57 32
i4 3 1 60 100
i4 5 1 62 100
i4 7 1 64 100
i4 9 1 65 100
i4 11 1 67 100
i4 13 1 69 100
i4 15 3 71 100
```

i4 18 3 72 100  
e

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

# vstnote

vstnote — Sends a MIDI note with definite duration to a VST plugin.

## Syntax

**vstnote** instance, kchan, knote, kveloc, kdur

## Description

*vstnote* sends a MIDI note with definite duration to a VST plugin.

## Initialization

*instance* - the number which identifies the plugin generated by *vstinit*.

## Performance

*kchan* - The MIDI channel to transmit the note on. Note that MIDI channels are numbered starting from 0.

*knote* - The MIDI note number to send.

*kveloc* - The MIDI note's velocity.

*kdur* - The MIDI note's duration in seconds.

Note: Be sure the instrument containing vstnote is not finished before the duration of the note, otherwise you'll have a 'hung' note.

## Examples

Here is an example of the use of the *vstnote* opcode. It uses the file *vst4cs.csd* [examples/vst4cs.csd].

### Exemple 588. Example of the *vstnote* opcode

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Credits: Adapted by Michael Gogins
; from code by David Horowitz and Lian Cheung.
; The "--displays" option is required in order for
; the Pianoteq GUI to dispatch events and display properly.
-m3 --displays -odac
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps   = 20
nchnls  = 2
; Load the Pianoteq into memory.
gipianoteq vstinit "C:\\Program Files\\Steinberg\\VstPlugins\\Pianoteq 3.0 Trial\\Pianoteq
```

```

; Print information about the Pianoteq, such as parameter names and numbers.
vstinfo      gipianoteq

; Open the Pianoteq's GUI.
vstedit      gipianoteq

; Send notes from the score to the Pianoteq.
instr 1
; MIDI channels are numbered starting at 0.
; p3 always contains the duration of the note.
; p4 contains the MIDI key number (pitch),
; p5 contains the MIDI velocity number (loudness),
imidichannel init      0
vstnote      gipianoteq, imidichannel, p4, p5, p3
endin

; Send parameter changes to the Pianoteq.
instr 2
; p4 is the parameter number.
; p5 is the parameter value.
vstparamset  gipianoteq, p4, p5
endin

; Send audio from the Pianoteq to the output.
instr 3
ablaninput   init      0
aleft, aright vstaudio  gipianoteq, ablaninput, ablaninput
outs         aleft, aright
endin

</CsInstruments>
<CsScore>
; Turn on the instrument that receives audio from the Pianoteq indefinitely.
i 3 0 -1
; Send parameter changes to Pianoteq before sending any notes.
; NOTE: All parameters must be between 0.0 and 1.0.
; Length of piano strings:
i 2 0 1 33 0.5
; Hammer noise:
i 2 0 1 25 0.1
; Send a C major 7th arpeggio to the Pianoteq.
i 1 1 10 60 76
i 1 2 10 64 73
i 1 3 10 67 70
i 1 4 10 71 67
; End the performance, leaving some time
; for the Pianoteq to finish sending out its audio,
; or for the user to play with the Pianoteq virtual keyboard.
e 20
</CsScore>
</CsSoundSynthesizer>

```

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

# vstparamset, vstparamget

vstparamset — Used for parameter communication to and from a VST plugin.

## Syntax

```
vstparamset instance, kparam, kvalue
```

```
kvalue vstparamget instance, kparam
```

## Description

*vstparamset* and *vstparamget* are used for parameter communication to and from a VST plugin.

## Initialization

*instance* - the number which identifies the plugin, to be passed to other vst4cs opcodes.

## Performance

*kparam* - The number of the parameter to set or get.

*kvalue* - the value to set, or the the value returned by the plugin.

Parameters vary according to the plugin. To find out what parameters are available, use the verbose option when loading the plugin with vstinit. Note that the VST protocol specifies that all parameter values must lie between 0.0 and 1.0, inclusive.

## Examples

Here is an example of the use of the *vstparamset* opcode. It uses the file *vst4cs.csd* [examples/vst4cs.csd].

### Exemple 589. Example of the *vstparamset* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Credits: Adapted by Michael Gogins
; from code by David Horowitz and Lian Cheung.
; The "--displays" option is required in order for
; the Pianoteq GUI to dispatch events and display properly.
-m3 --displays -odac
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps   = 20
nchnls  = 2
gipianoteq      ; Load the Pianoteq into memory.
vstinit         "C:\\Program Files\\Steinberg\\VstPlugins\\Pianoteq 3.0 Trial\\Pianoteq
; Print information about the Pianoteq, such as parameter names and numbers.
```

```

vstinfo          gipianoteq

; Open the Pianoteq's GUI.
vstedit          gipianoteq

; Send notes from the score to the Pianoteq.
instr 1
; MIDI channels are numbered starting at 0.
; p3 always contains the duration of the note.
; p4 contains the MIDI key number (pitch),
; p5 contains the MIDI velocity number (loudness),
imidichannel     init          0
vstnote          gipianoteq, imidichannel, p4, p5, p3
endin

; Send parameter changes to the Pianoteq.
instr 2
; p4 is the parameter number.
; p5 is the parameter value.
vstparamset      gipianoteq, p4, p5
endin

; Send audio from the Pianoteq to the output.
instr 3
ablankinput      init          0
aleft, aright    vstaudio      gipianoteq, ablankinput, ablankinput
outs             aleft, aright
endin

</CsInstruments>
<CsScore>
; Turn on the instrument that receives audio from the Pianoteq indefinitely.
i 3 0 -1
; Send parameter changes to Pianoteq before sending any notes.
; NOTE: All parameters must be between 0.0 and 1.0.
; Length of piano strings:
i 2 0 1 33 0.5
; Hammer noise:
i 2 0 1 25 0.1
; Send a C major 7th arpeggio to the Pianoteq.
i 1 1 10 60 76
i 1 2 10 64 73
i 1 3 10 67 70
i 1 4 10 71 67
; End the performance, leaving some time
; for the Pianoteq to finish sending out its audio,
; or for the user to play with the Pianoteq virtual keyboard.
e 20
</CsScore>
</CsoundSynthesizer>

```

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

# vstprogset

vstprogset — Loads parameter banks to a VST plugin.

## Syntax

```
vstprogset instance, kprogram
```

## Description

*vstprogset* sets one of the programs in an `.fxb` bank.

## Initialization

*instance* - the number which identifies the plugin, to be passed to other vst4cs opcodes.

*kprogram* - the number of the program to set.

## Examples

### Exemple 590. Usage of vstprogset

```
/* orc */
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
giHandle1 vstinit "c:/vstplugins/cheeze/cheeze machine.dll",1
instr 4
vstbankload gihandle1,"c:/vstplugins/cheeze/chengo'scheese.fxb"
vstprogset gihandle1, 4
vstinfo gihandle1
endin

/* sco */
i 3 0 21
i4 1 1 57 32
e
```

## Credits

By: Andrés Cabrera and Michael Gogins

Uses code from Hermann Seib's VSTHost and Thomas Grill's vst~ object.

VST is a trademark of Steinberg Media Technologies GmbH. VST Plug-In Technology by Steinberg.

# vsubv

vsubv — Performs subtraction between two vectorial control signals

## Description

Performs subtraction between two vectorial control signals

## Syntax

```
vsubv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

## Performance

*kelements* - number of elements of the two vectors

*kdstoffset* - index offset for the destination (ifn1) table (Default=0)

*ksrcoffset* - index offset for the source (ifn2) table (Default=0)

*kverbose* - Selects whether or not warnings are printed (Default=0)

*vsubv* subtracts two vectorial control signals, that is, each element of *ifn2* is subtracted from the corresponding element of *ifn1*. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *kdstoffset* and *ksrcoffset* to specify vectors in any location of the tables.

Negative values for *kdstoffset* and *ksrcoffset* are acceptable. If *kdstoffset* is negative, the out of range section of the vector will be discarded. If *ksrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector will not be changed for these elements).

If the optional *kverbose* argument is different to 0, the opcode will print warning messages every k-pass if table lengths are exceeded.



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at k-rate (this means that every k-pass the vectors are subtracted). There's an i-rate



version of this opcode called *vsubv\_i*.



## Note

Please note that the *elements* argument has changed in version 5.03 from i-rate to k-rate. This will change the opcode's behavior in the unusual cases where the i-rate variable *ielements* is changed inside the instrument, for example in:

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

All these operators (*vaddy*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Examples

Here is an example of the *vsubv* opcode. It uses the file *vsubv.csd* [examples/vsubv.csd].

### Exemple 591. Example of the *vsubv* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vsubv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
```

```
        endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16
f 2 0 16 -7 1 15 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vsubv\_i

vsubv\_i — Performs subtraction between two vectorial control signals at init time.

## Description

Performs subtraction between two vectorial control signals at init time.

## Syntax

```
vsubv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialization

*ifn1* - number of the table hosting the first vector to be processed

*ifn2* - number of the table hosting the second vector to be processed

*ielements* - number of elements of the two vectors

*idstoffset* - index offset for the destination (*ifn1*) table (Default=0)

*isrcoffset* - index offset for the source (*ifn2*) table (Default=0)

## Performance

*vsubv\_i* subtracts two vectorial control signals, that is, each element of *ifn2* is subtracted from the corresponding element of *ifn1*. Each vectorial signal is hosted by a table (*ifn1* and *ifn2*). The number of elements contained in both vectors must be the same.

The result is a new vectorial control signal that overrides old values of *ifn1*. If you want to keep the old *ifn1* vector, use *vcopy\_i* opcode to copy it in another table. You can use *idstoffset* and *isrcoffset* to specify vectors in any location of the tables.

Negative values for *idstoffset* and *isrcoffset* are acceptable. If *idstoffset* is negative, the out of range section of the vector will be discarded. If *isrcoffset* is negative, the out of range elements will be assumed to be 0 (i.e. the destination elements will not be changed). If elements for the destination vector are beyond the size of the table (including guard point), these elements are discarded (i.e. elements do not wrap around the tables). If elements for the source vector are beyond the table length, these elements are taken as 0 (i.e. the destination vector will not be changed for these elements).



### Avertissement

Using the same table as source and destination table in versions earlier than 5.04, might produce unexpected behavior, so use with care.

This opcode works at init time. There's an k-rate version of this opcode called *vsubv*.

All these operators (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* and *vmap*) are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

## Credits

Written by Gabriel Maldonado. Optional arguments added by Andres Cabrera and Istvan Varga.

New in Csound 5 (Previously available only on CsoundAV)

# vtable1k

vtable1k — Read a vector (several scalars simultaneously) from a table.

## Description

This opcode reads vectors from tables at k-rate.

## Syntax

```
vtable1k kfn,kout1 [, kout2, kout3, .... , koutN ]
```

## Performance

*kfn* - table number

*kout1...koutN* - output vector elements

*vtable1k* is a reduced version of *vtablek*, it only allows to access the first vector (it is equivalent to *vtablek* with *kndx* = zero, but a bit faster). It is useful to easily and quickly convert a set of values stored in a table into a set of k-rate variables to be used in normal opcodes, instead of using individual *table* opcodes for each value.



### Note

*vtable1k* is an unusual opcode as it produces its output on the right side arguments of the opcode.

## Examples

Here is an example of the *vtable1k* opcode. It uses the files *vtable1k.csd* [examples/vtable1k.csd].

### Exemple 592. Example of the vtable1k opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc     -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 100
nchnls = 2

giElem init 13
giOutTab ftgen 1,0,128, 2,          0
giFreqTab ftgen 2,0,128,-7,        1,giElem, giElem+1
giSine ftgen 3,0,256,10, 1

FLpanel "This Panel contains a Slider Bank",500,400
FLslidBnk "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab, 360, 600, 100, 10
FLpanel_end
```

```

    FLrun

    instr 1

    kout1 init 0
    kout2 init 0
    kout3 init 0
    kout4 init 0
    kout5 init 0
    kout6 init 0
    kout7 init 0
    kout8 init 0

    vtablelk giOutTab, kout1 , kout2, kout3, kout4, kout5 , kout6, kout7, kout8
    kmodindex1= 2 * db(kout1 * 80 )
    kmodindex2= 2 * db(kout2 * 80 )
    kmodindex3= 2 * db(kout3 * 80 )
    kamp = 50 * db(kout4 * 70 )
    kfreq1 = 1.1 * octave(kout5 * 10)
    kfreq2 = 1.1 * octave(kout6 * 10)
    kfreq3 = 1.1 * octave(kout7 * 10)
    kfreq4 = 30 * octave(kout8 * 8)

    amod1 oscili kmodindex1, kfreq1, giSine
    amod2 oscili kmodindex2, kfreq2, giSine
    amod3 oscili kmodindex3, kfreq3, giSine
    aout oscili kamp, kfreq4+amod1+amod2+amod3, giSine

    outs aout, aout
    endin

</CsInstruments>
<CsScore>

i1 0 3600
f0 3600

</CsScore>
</CsoundSynthesizer>

```

## See Also

*vtablek*

## Credits

Written by Gabriel Maldonado.

New in Csound 5.06

# vtablei

vtablei — Read vectors (from tables -or arrays of vectors).

## Description

This opcode reads vectors from tables.

## Syntax

```
vtablei  indx, ifn, interp, ixmode, iout1 [ , iout2, iout3, .... , ioutN ]
```

## Initialization

*indx* - Index into f-table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0).

*ifn* - table number

*iout1...ioutN* - output vector elements

*ixmode* - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

*interp* - vtable (vector table) family of opcodes allows the user to switch between interpolated or non-interpolated output by means of the *interp* argument.

## Performance

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (*iout1* , *iout2*, *iout3*, .... *ioutN*).

*vtable* (vector table) family of opcodes allows the user to switch between interpolated or non-interpolated output by means of the *interp* argument.

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtable*, in order to correct eventual out-of-range values.



### Note

Notice that *vtablei*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

## Examples

Here is an example of the `vtablei` opcode. It uses the files `vtablei.csd` [examples/vtablei.csd]

### Exemple 593. Example of the `vtablei` opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      100
ksmps   =      441
nchnls  =      2

gindx init 0

      instr 1
kindex init 0
ktrig metro 0.5
if ktrig = 0 goto noevent
event "i", 2, 0, 0.5, kindex
kindex = kindex + 1
noevent:

      endin

      instr 2
iout1 init 0
iout2 init 0
iout3 init 0
iout4 init 0
indx = p4
vtablei indx, 1, 1, 0, iout1,iout2, iout3, iout4
print iout1, iout2, iout3, iout4
turnoff
      endin

</CsInstruments>
<CsScore>
f 1 0 32 10 1
i 1 0 20

</CsScore>
</CsoundSynthesizer>
```

## See also

`vtablea`, `vtablek`, `vtabi`, `vtablewi`, `vtabwi`,

## Credits

Example written by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)



# vtablek

vtablek — Read vectors (from tables -or arrays of vectors).

## Description

This opcode reads vectors from tables at k-rate.

## Syntax

```
vtablek kndx, kfn, kinterp, ixmode, kout1 [, kout2, kout3, .... , koutN ]
```

## Initialization

*ixmode* - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

*kinterp* - switch between interpolated or non-interpolated output. 0 -> non-interpolation , non-zero -> interpolation activated

## Performance

*kndx* - Index into f-table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0).

*kfn* - table number

*kout1...koutN* - output vector elements

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (*kout1* , *kout2*, *kout3*, .... *koutN*).

*vtablek* allows the user to switch between interpolated or non-interpolated output at k-rate by means of *kinterp* argument.

*vtablek* allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occur), as well as to choose indexing style (raw or normalized, see also *ixmode* argument of table opcode ).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtable*, in order to correct eventual out-of-range values.



### Note

Notice that *vtablek*'s output arguments are placed at the left of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output argu-

ments such as *fin* or *trigseq*).

## Examples

Here is an example of the *vtablek* opcode. It uses the files *vtablek.csd* [examples/vtablek.csd].

### Exemple 594. Example of the *vtablek* opcode.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

  sr      =      44100
  kr      =      100
  ksmpr    =      441
  nchnls  =      2

  gkindx  init  -1

      instr  1
kindex  init  0
ktrig  metro  0.5
if ktrig = 0 goto noevent
gkindx = gkindx + 1
noevent:

      endin

      instr  2
kout1  init  0
kout2  init  0
kout3  init  0
kout4  init  0
vtablek  gkindx, 1, 1, 0, kout1,kout2, kout3, kout4
printk2  kout1
printk2  kout2
printk2  kout3
printk2  kout4
      endin

</CsInstruments>
<CsScore>
f 1 0 32 10 1
i 1 0 20
i 2 0 20
</CsScore>
</CsoundSynthesizer>
```

## See also

*vtablea*, *vtablei*, *vtabk*, *vtablewk*, *vtabwk*,

## Credits

Written by Gabriel Maldonado.

Example written by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# vtablea

vtablea — Read vectors (from tables -or arrays of vectors).

## Description

This opcode reads vectors from tables at a-rate.

## Syntax

```
vtablea andx, kfn, kinterp, ixmode, aout1 [, aout2, aout3, .... , aoutN ]
```

## Initialization

*ixmode* - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

## Performance

*andx* - Index into f-table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0).

*kfn* - table number

*kinterp* - switch between interpolated or non-interpolated output. 0 -> non-interpolation , non-zero -> interpolation activated

*aout1...aoutN* - output vector elements

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (*aout1* , *aout2*, *aout3*, .... *aoutN*).

**vtablea** allows the user to switch between interpolated or non-interpolated output at k-rate by means of *kinterp* argument.

**vtablea** allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occur), as well as to choose indexing style (raw or normalized, see also *ixmode* argument of table opcode ).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using **vtablea**, in order to correct eventual out-of-range values.



### Note

Notice that *vtablea*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output

arguments such as *fin* or *trigseq*).

## See also

*vtablek*, *vtablei*, *vtaba*, *vtablewa*, *vtabwa*,

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vtablewi

vtablewi — Write vectors (to tables -or arrays of vectors).

## Description

This opcode writes vectors to tables at init time.

## Syntax

```
vtablewi  indx, ifn, ixmode, inarg1 [ , inarg2, inarg3 , .... , inargN ]
```

## Initialization

*indx* - Index into f-table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0).

*ifn* - table number

*ixmode* - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

*inarg1...inargN* - output vector elements

## Performance

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (*inarg1* , *inarg2*, *inarg3*, .... *inargN*).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtablewi*, in order to correct eventual out-of-range values.

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vtablewk

vtablewk — Write vectors (to tables -or arrays of vectors).

## Description

This opcode writes vectors to tables at k-rate.

## Syntax

```
vtablewk kndx, kfn, ixmode, kinarg1 [ , kinarg2, kinarg3 , .... , kinargN ]
```

## Initialization

*ixmode* - index data mode. The default value is 0. == 0 index is treated as a raw table location, == 1 index is normalized (0 to 1).

## Performance

*kndx* - Index into f-table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0).

*kfn* - table number

*kinarg1...kinargN* - output vector elements

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (*kinarg1* , *kinarg2*, *kinarg3*, .... *kinargN*).

**vtablewk** allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occur), as well as to choose indexing style (raw or normalized, see also *ixmode* argument of table opcode ).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtablewk*, in order to correct eventual out-of-range values.

## Examples

Here is an example of the *vtablewk* opcode. It uses the files *vtablewk.csd* [examples/vtablewk.csd].

### Exemple 595. Example of the vtablewk opcode.

```
<CsoundSynthesizer>
<CsOptions>
;-ovtablewa.wav -W -b441 -B441
-odac -b441 -B441
</CsOptions>
```

```

<CsInstruments>

sr=44100
kr=441
ksmps=100
nchnls=2

    instr 1
ilen = ftlen (1)

knew1 oscil 10000, 440, 3
knew2 oscil 15000, 440, 3, 0.5
kindex phasor 0.3
asig oscil 1, sr/ilen , 1
vtablewk kindex*ilen, 1, 0, knew1, knew2
out asig,asig
    endin

</CsInstruments>
<CsScore>
f1 0 262144 -1 "beats.aiff" 0 4 0
f2 0 262144 2 0
f3 0 1024 10 1

i1 0 10
</CsScore>
</CsoundSynthesizer>

```

## Credits

Written by Gabriel Maldonado.

Example written by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# vtablewa

vtablewa — Write vectors (to tables -or arrays of vectors).

## Description

This opcode writes vectors to tables at a-rate.

## Syntax

```
vtablewa andx, kfn, ixmode, ainarg1 [ , ainarg2, ainarg3 , .... , ainargN ]
```

## Initialization

*ixmode* - index data mode. The default value is 0.

== 0 index is treated as a raw table location,

== 1 index is normalized (0 to 1).

## Performance

*andx* - Index into f-table, either a positive number range matching the table length (*ixmode* = 0) or a 0 to 1 range (*ixmode* != 0).

*kfn* - table number

*ainarg1...ainargN* - input vector elements

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (*ainarg1* , *ainarg2*, *ainarg3*, .... *ainargN*).

*vtablewa* allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occur), as well as to choose indexing style (raw or normalized, see also *ixmode* argument of table opcode ).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtablewa*, in order to correct eventual out-of-range values.

## Examples

Here is an example of the *vtablewa* opcode. It uses the files *vtablewa.csd* [examples/vtablewa.csd].

### Exemple 596. Example of the vtablewa opcode.

```
<CsoundSynthesizer>  
<CsOptions>
```



```
-odac -b441 -B441
</CsOptions>
<CsInstruments>

  sr=44100
  kr=4410
  ksmps=10
  nchnls=2

  instr 1
  vcopy
  ar random 0, 1
  vtablewa ar
  out ar,ar
  endin

</CsInstruments>
<CsScore>
f1 0 262144 -1 "beats.wav" 0 4 0
f2 0 262144 2 0

i1 0 4
i2 3 1

s
i1 0 4
i3 3 1
s

i1 0 4

</CsScore>
</CsoundSynthesizer>
```

## Credits

Written by Gabriel Maldonado.

Example written by Andres Cabrera.

New in Csound 5 (Previously available only on CsoundAV)

# vtabi

vtabi — Read vectors (from tables -or arrays of vectors).

## Description

This opcode reads vectors from tables.

## Syntax

```
vtabi  indx, ifn, iout1 [, iout2, iout3, .... , ioutN ]
```

## Initialization

*indx* - Index into f-table, either a positive number range matching the table length

*ifn* - table number

*iout1...ioutN* - output vector elements

## Performance

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (iout1 , iout2, iout3, .... ioutN).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using vtable, in order to correct eventual out-of-range values.

Notice that vtabi output arguments are placed at the left of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as fin or trigseq).

The **vtab** family is similar to **vtable**, but is much faster because interpolation is not available, table number cannot be changed after initialization, and only raw indexing is supported.



### Note

Notice that *vtabi*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

## Examples

For an example of the vtabi opcode usage, see *vtablei*.

## See also

*vtabk, vtaba, vtablei, vtablewi, vtabwi,*

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vtabk

vtabk — Read vectors (from tables -or arrays of vectors).

## Description

This opcode reads vectors from tables at k-rate.

## Syntax

```
vtabk kndx, ifn, kout1 [, kout2, kout3, .... , koutN ]
```

## Initialization

*ifn* - table number

## Performance

*kndx* - Index into f-table, either a positive number range matching the table length

*kout1...koutN* - output vector elements

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.) . The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (*kout1* , *kout2*, *kout3*, .... *koutN*).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtable*, in order to correct eventual out-of-range values.

Notice that *vtabk* output arguments are placed at the left of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

The **vtab** family is similar to **vtable**, but is much faster because interpolation is not available, table number cannot be changed after initialization, and only raw indexing is supported.



### Note

Notice that *vtabk*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

## Examples

For an example of the *vtabk* opcode usage, see *vtablek*.

## See also

*vtabi, vtaba, vtablek, vtablewk, vtabwk,*

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vtaba

vtaba — Read vectors (from tables -or arrays of vectors).

## Description

This opcode reads vectors from tables at a-rate.

## Syntax

```
vtaba andx, ifn, aout1 [, aout2, aout3, .... , aoutN ]
```

## Initialization

*ifn* - table number

## Performance

*andx* - Index into f-table, either a positive number range matching the table length

*aout1...aoutN* - output vector elements

This opcode is useful in all cases in which one needs to access sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (aout1 , aout2, aout3, .... aoutN).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using vtaba, in order to correct eventual out-of-range values.

Notice that **vtaba** output arguments are placed at the left of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

The **vtab** family is similar to the **vtable** family, but is much faster because interpolation is not available, table number cannot be changed after initialization, and only raw indexing is supported.



### Note

Notice that *vtaba*'s output arguments are placed at the right of the opcode name, differently from usual (this style is already used in other opcodes using undefined lists of output arguments such as *fin* or *trigseq*).

## Examples

The usage of *vtaba* is similar to *vtablek*.

## See also

*vtabk, vtabi, vtablea, vtablewa, vtabwa,*

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vtabwi

vtabwi — Write vectors (to tables -or arrays of vectors).

## Description

This opcode writes vectors to tables at init time.

## Syntax

```
vtabwi  indx, ifn, inarg1 [, inarg2, inarg3 , .... , inargN ]
```

## Initialization

*indx* - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

*ifn* - table number

*inarg1...inargN* - output vector elements

## Performance

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (inarg1 , inarg2, inarg3, .... inargN).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtabwi*, in order to correct eventual out-of-range values.

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)



# **vtabwk**

vtabwk — Write vectors (to tables -or arrays of vectors).

## **Description**

This opcode writes vectors to tables at a-rate.

## **Syntax**

```
vtabwk kndx, ifn, kinarg1 [ , kinarg2, kinarg3 , .... , kinargN ]
```

## **Initialization**

*ifn* - table number

## **Performance**

*kndx* - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0). *kinarg1...kinargN* - input vector elements

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.) . The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (kinarg1 , kinarg2, kinarg3, .... kinargN).

*vtabwk* allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occur), as well as to choose indexing style (raw or normalized, see also ixmode argument of table opcode ).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtabwk*, in order to correct eventual out-of-range values.

## **Credits**

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# **vtabwa**

vtabwa — Write vectors (to tables -or arrays of vectors).

## **Description**

This opcode writes vectors to tables at a-rate.

## **Syntax**

```
vtabwa  andx, ifn, ainarg1 [ , ainarg2, ainarg3 , .... , ainargN ]
```

## **Initialization**

*ifn* - table number

## **Performance**

*andx* - Index into f-table, either a positive number range matching the table length (ixmode = 0) or a 0 to 1 range (ixmode != 0).

*ainarg1...ainargN* - input vector elements

This opcode is useful in all cases in which one needs to write sets of values associated to unique indexes (for example, multi-channel samples, STFT bin frames, spectral formants, p-field based scores etc.). The number of elements of each vector (length of the vector) is determined by the number of optional arguments on the right (ainarg1 , ainarg2, ainarg3, .... ainargN).

*vtabwa* allows also to switch the table number at k-rate (but this is possible only when vector frames of each used table have the same number of elements, otherwise unpredictable results could occur), as well as to choose indexing style (raw or normalized, see also ixmode argument of table opcode ).

Notice that no wrap nor limit mode is implemented. So, if an index attempt to access to a zone not allocated by the table, Csound will probably crash. However this drawback can be easily avoided by using wrap or limit opcodes applied to indexes before using *vtabwa*, in order to correct eventual out-of-range values.

## **Credits**

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# vwrap

vwrap — Limiting and Wrapping Vectorial Signals

## Description

Wraps elements of vectorial control signals.

## Syntax

```
vwrap ifn, kmin, kmax, ielements
```

## Initialization

*ifn* - number of the table hosting the vector to be processed

*ielements* - number of elements of the vector

## Performance

*kmin* - minimum threshold value

*kmax* - maximum threshold value

*vwrap* wraps around each element of corresponding vector if it exceeds low or high thresholds.

These opcodes are similar to *limit*, *wrap* and *mirror*, but operate with a vectorial signal instead of with a scalar signal.

Result overrides old values of *ifn1*, if these are out of min/max interval. If you want to keep input vector, use *vcopy* opcode to copy it in another table.

All these opcodes are designed to be used together with other opcodes that operate with vectorial signals such as *bmscan*, *vcella*, *adsynt*, *adsynt2* etc.

*Note:* *bmscan* not yet available on Canonical Csound

## Credits

Written by Gabriel Maldonado.

New in Csound 5 (Previously available only on CsoundAV)

# waveset

waveset — Un variateur de durée simple par répétition de périodes.

## Description

Un variateur de durée simple par répétition de périodes.

## Syntaxe

```
ares waveset ain, krep [, ilen]
```

## Initialisation

*ilen* (facultatif, 0 par défaut) -- la longueur (en échantillons) du signal audio. Si *ilen* vaut 0, la moitié de la longueur de la note donnée (p3) est prise.

## Exécution

*ain* -- le signal audio en entrée.

*krep* -- le nombre de fois que la période est répétée.

L'entrée est lue et chaque période complète (deux passages par zéro) est répétée *krep* fois.

Il y a un tampon interne car la sortie est évidemment plus lente que l'entrée. Il faut faire attention si le tampon est trop court, car il peut y avoir des effets étranges.

## Exemples

Voici un exemple de l'opcode waveset. Il utilise les fichiers *waveset.csd* [examples/waveset.csd] et *beats.wav* [examples/beats.wav].

### Exemple 597. Exemple de l'opcode waveset.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o waveset.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1 - play an audio file.
instr 1
  asig soundin "beats.wav"
  out asig
endin

; Instrument #2 - stretch the audio file with waveset.
instr 2
  asig soundin "beats.wav"
  al waveset asig, 2

  out al
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for four seconds.
i 2 3 4
e

</CsScore>
</CsSoundSynthesizer>
```

## Crédits

Auteur : John ffitich  
Février 2001

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.11

# weibull

weibull — Générateur de nombres aléatoires de distribution de Weibull (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution de Weibull (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares weibull ksigma, ktau
```

```
ires weibull ksigma, ktau
```

```
kres weibull ksigma, ktau
```

## Exécution

*ksigma* -- contrôle l'échelle de l'étalement de la distribution.

*ktau* -- s'il est supérieur à un, les nombres proches de *ksigma* sont favorisés. S'il est inférieur à un, les petites valeurs sont favorisées. S'il est égal à 1, la distribution est exponentielle. Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode weibull. Il utilise le fichier *weibull.csd* [examples/weibull.csd].

### Exemple 598. Exemple de l'opcode weibull.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o weibull.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a random number in a Weibull distribution.
; ksigma = 1
; ktau = 1

i1 weibull 1, 1

print i1
endin

; Instrument #2.
instr 2
; Generate a random number in a Weibull distribution.
; ksigma = 1
; ktau = 1

seed 0

i1 weibull 1, 1

print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 1 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1: i1 = 1.834
```

## Voir Aussi

*seed, betarand, bexpnd, cauchy, exprand, gauss, linrand, pcauchy, poisson, trirand, unirand*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Exemple écrit par Kevin Conder.

# wgbow

wgbow — Simule un son de corde frottée.

## Description

La sortie audio simule un son de corde frottée, réalisé au moyen d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

ares **wgbow** kamp, kfreq, kpres, krat, kvibf, kvamp, ifn [, iminfreq]

## Initialisation

*ifn* -- table contenant la forme du vibrato, habituellement une table de sinusoïde, créée par une fonction.

*iminfreq* (facultatif) -- fréquence la plus grave à laquelle l'instrument sera joué. Si elle est omise, elle prend la valeur initiale de *kfreq*. Si *iminfreq* est négative, l'initialisation est ignorée.

## Exécution

Une note est jouée sur un instrument de type corde, avec les arguments ci-dessous.

*kamp* -- amplitude de la note.

*kfreq* -- fréquence de la note jouée.

*kpres* -- un paramètre contrôlant la pression de l'archet sur la corde. Les valeurs doivent se situer autour de 3. L'intervalle utile va approximativement de 1 à 5.

*krat* -- la position de l'archet le long de la corde. Le jeu habituel se fait environ à 0.127236. L'intervalle recommandé va de 0.025 à 0.23.

*kvibf* -- fréquence du vibrato en Hz. L'intervalle recommandé va de 0 à 12.

*kvamp* -- l'amplitude du vibrato.

## Exemples

Voici un exemple de l'opcode wgbow. Il utilise le fichier *wgbow.csd* [examples/wgbow.csd].

### Exemple 599. Exemple de l'opcode wgbow.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```



```

-odac          -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgbow.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 31129.60
  kfreq = 440
  kpres = 3.0
  krat = 0.127236
  kvibf = 6.12723
  ifn = 1

  ; Create an amplitude envelope for the vibrato.
  kv linseg 0, 0.5, 0, 1, 1, p3-0.5, 1
  kvamp = kv * 0.01

  al wgbow kamp, kfreq, kpres, krat, kvibf, kvamp, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 128 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# wgbowedbar

wgbowedbar — Modèle physique d'une barre frottée.

## Description

Modèle physique d'une barre frottée, appartenant à la famille des instruments à guide d'onde de Perry Cook.

## Syntaxe

```
ares wgbowedbar kamp, kfreq, kpos, kbowpres, kgain [, iconst] [, itvel] \  
    [, ibowpos] [, ilow]
```

## Initialisation

*iconst* (facultatif, 0 par défaut) -- une constante d'intégration. Vaut zéro par défaut.

*itvel* (facultatif, 0 par défaut) -- 0 ou 1. Quand *itvel* = 0, la vitesse de l'archet suit une trajectoire de type ADSR. Quand *itvel* = 1, la valeur de la vélocité de l'archet décroît exponentiellement.

*ibowpos* (facultatif, 0 par défaut) -- la position sur l'archet, qui affecte la trajectoire de vélocité de l'archet.

*ilow* (facultatif, 0 par défaut) -- fréquence la plus basse désirée.

## Exécution

*kamp* -- amplitude du signal.

*kfreq* -- fréquence du signal.

*kpos* -- position de l'archet sur la barre, comprise entre 0 et 1.

*kbowpres* -- pression de l'archet (comme dans *wgbowed*)

*kgain* -- gain du filtre. On recommande une valeur d'environ 0.809.

## Exemples

Voici un exemple de l'opcode *wgbowedbar*. Il utilise le fichier *wgbowedbar.csd* [examples/wgbowedbar.csd].

### Exemple 600. Exemple de l'opcode *wgbowedbar*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in   No messages
-odac        -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgbowedbar.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
; pos = [0, 1]
; bowpress = [1, 10]
; gain = [0.8, 1]
; intr = [0, 1]
; trackvel = [0, 1]
; bowpos = [0, 1]

kb line 0.5, p3, 0.1
kp line 0.6, p3, 0.7
kc line 1, p3, 1

a1 wgbowedbar p4, cpspch(p5), kb, kp, 0.995, p6, 0

out a1
endin

</CsInstruments>
<CsScore>

i1 0 3 32000 7.00 0
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 4.07 de Csound

# wgbrass

wgbrass — Simule un son de cuivre.

## Description

La sortie audio simule un son de cuivre, réalisé au moyen d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp, ifn [, iminfreq]
```

## Initialisation

*iatt* -- temps requis pour atteindre la pression nominale.

*ifn* -- table contenant la forme du vibrato, habituellement une table de sinusoïde, créée par une fonction

*iminfreq* -- (facultatif) -- fréquence la plus grave à laquelle l'instrument sera joué. Si elle est omise, elle prend la valeur initiale de *kfreq*. Si *iminfreq* est négative, l'initialisation est ignorée.

## Exécution

Une note est jouée sur un instrument de type cuivre, avec les arguments ci-dessous.

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*ktens* -- Tension des lèvres de l'instrumentiste. La valeur recommandée vaut environ 0.4.

*kvibf* -- Fréquence du vibrato en Hz. L'intervalle recommandé va de 0 à 12.

*kvamp* -- amplitude du vibrato



### NOTE

Ceci est assez pauvre et non contrôlable. Il faut une révision, et probablement plus de paramètres.

## Exemples

Voici un exemple de l'opcode wgbrass. Il utilise le fichier *wgbrass.csd* [exemples/wgbrass.csd].

### Exemple 601. Exemple de l'opcode wgbrass.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o wgbrass.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1
0dbfs = 1

; Instrument #1.
instr 1
  kamp = 0.7
  kfreq = p4
  ktens = p5
  iatt = p6
  kvibf = p7
  ifn = 1

  ; Create an amplitude envelope for the vibrato.
  kvamp line 0, p3, 0.5

  al wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 1024 10 1

;
i 1 0 4 440 0.4 0.1 6.137
i 1 4 4 440 0.4 0.01 0.137
i 1 8 4 880 0.4 0.1 6.137
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# wgclar

wgclar — Simule un son de clarinette.

## Description

La sortie audio simule un son de clarinette, réalisé au moyen d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares wgclar kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, ifn \
    [, iminfreq]
```

## Initialisation

*iatt* -- temps en secondes nécessaire pour atteindre la pression de souffle nominale. 0.1 semble correspondre à un jeu raisonnable. Une durée plus longue donne un son initial de vent défini.

*idetk* -- temps en secondes pour arrêter le souffle. 0.1 correspond à une extinction douce.

*ifn* -- table contenant la forme du vibrato, habituellement une table de sinusoïde, créée par une fonction

*iminfreq* (facultatif) -- fréquence la plus grave à laquelle l'instrument sera joué. Si elle est omise, elle prend la valeur initiale de *kfreq*. Si *iminfreq* est négative, l'initialisation est ignorée.

## Exécution

Une note est jouée sur un instrument de type clarinette, avec les arguments ci-dessous.

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kstiff* -- Paramètre de raideur de l'anche. Les valeurs doivent être négatives, aux environs de -0.3. L'intervalle utile est compris approximativement entre -0.44 et -0.18.

*kngain* -- Amplitude de la composante de bruit, approximativement comprise entre 0 et 0.5.

*kvibf* -- Fréquence du vibrato en Hz. L'intervalle recommandé va de 0 à 12.

*kvamp* -- Amplitude du vibrato

## Exemples

Voici un exemple de l'opcode wgclar. Il utilise le fichier *wgclar.csd* [examples/wgclar.csd].

### Exemple 602. Exemple de l'opcode wgclar.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgclar.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp init 31129.60
  kfreq = 440
  kstiff = -0.3
  iatt = 0.1
  idetk = 0.1
  kngain = 0.2
  kvibf = 5.735
  kvamp = 0.1
  ifn = 1

  al wgclar kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitc (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# wgflute

wgflute — Simule un son de flûte.

## Description

La sortie audio simule un son de flûte, réalisé au moyen d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares wgflute kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, ifn \
    [, iminfreq] [, ijetrf] [, iendrf]
```

## Initialisation

*iatt* -- temps en secondes nécessaire pour atteindre la pression de souffle nominale. 0.1 semble correspondre à un jeu raisonnable.

*idetk* -- temps en secondes pour arrêter le souffle. 0.1 correspond à une extinction douce.

*ifn* -- table contenant la forme du vibrato, habituellement une table de sinusöide, créée par une fonction

*iminfreq* (facultatif) -- fréquence la plus grave à laquelle l'instrument sera joué. Si elle est omise, elle prend la valeur initiale de *kfreq*. Si *iminfreq* est négative, l'initialisation est ignorée.

*ijetrf* (facultatif, 0.5 par défaut) -- quantité de réflexion dans le jet d'air qui excite la flûte. La valeur par défaut est 0.5.

*iendrf* (facultatif, 0.5 par défaut) -- coefficient de réflexion du jet d'air. La valeur par défaut est 0.5. *ijetrf* et *iendrf* sont utilisés dans le calcul de la pression différentielle.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée. Elle peut être variée pendant l'exécution, mais je n'ai pas essayé.

*kjet* -- un paramètre contrôlant le jet d'air. Ses valeurs doivent être positives, aux environs de 0.3. L'intervalle utile est compris approximativement entre 0.08 et 0.56.

*kngain* -- amplitude de la composante de bruit, approximativement comprise entre 0 et 0.5.

*kvibf* -- fréquence du vibrato en Hz. L'intervalle recommandé va de 0 à 12

*kvamp* -- amplitude du vibrato

## Exemples

Voici un exemple de l'opcode wgflute. Il utilise le fichier *wgflute.csd* [examples/wgflute.csd].

### Exemple 603. Exemple de l'opcode wgflute.



Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgflute.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 31129.60
  kfreq = 440
  kjet = 0.32
  iatt = 0.1
  idetk = 0.1
  kngain = 0.15
  kvibf = 5.925
  kvamp = 0.05
  ifn = 1

  al wgflute kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# wgpluck

wgpluck — Une simulation haute fidélité de corde pincée.

## Description

Une simulation haute fidélité de corde pincée, utilisant des lignes à retard avec interpolation.

## Syntaxe

ares **wgpluck** *icps*, *iamp*, *kpick*, *iplk*, *idamp*, *ifilt*, *axcite*

## Initialisation

*icps* -- fréquence de la corde pincée

*iamp* -- amplitude de la corde pincée

*iplk* -- point d'excitation le long de la corde, dans l'intervalle compris entre 0 et 1. 0 = pas d'excitation.

*idamp* -- amortissement de la note. Il contrôle l'extinction globale de la corde. Plus la valeur de *idamp* est importante, plus la décroissance est rapide. Avec une valeur négative, il y aura un accroissement progressif de la sortie.

*ifilt* -- contrôle l'atténuation du filtre sur le chevalet. Les valeurs élevées provoquent une décroissance plus rapide des harmoniques supérieurs.

## Exécution

*kpick* -- Fraction de la longueur de la corde où sera lue la sortie.

*axcite* -- un signal d'excitation de la corde.

Une corde de fréquence *icps* est pincée avec l'amplitude *iamp* au point *iplk*. L'extinction de la corde virtuelle est contrôlée par *idamp* et *ifilt* qui simule le chevalet. L'oscillation est lue au point *kpick*, et excitée par le signal *axcite*.

## Exemples

L'exemple suivant produit une note moyennement longue avec une décroissance rapide des partiels supérieurs. Il utilise le fichier *wgpluck.csd* [examples/wgpluck.csd].

### Exemple 604. Un exemple de l'opcode wgpluck.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```

```

-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wGPLUCK.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  icps = 220
  iamp = 20000
  kpick = 0.5
  iplk = 0
  idamp = 10
  ifilt = 1000

  excite oscil 1, 1, 1
  apluck wGPLUCK icps, iamp, kpick, iplk, idamp, ifilt, excite

  out apluck
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

L'exemple suivant produit une note plus courte et plus brillante. Il utilise le fichier file *wGPLUCK\_brighter.csd* [examples/wGPLUCK\_brighter.csd].

### Exemple 605. Un exemple de l'opcode *wGPLUCK* avec une note plus courte et plus brillante.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wGPLUCK_brighter.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  icps = 220
  iamp = 20000
  kpick = 0.5
  iplk = 0
  idamp = 30
  ifilt = 10

```

```
axcite oscil 1, 1, 1
apluck wgpluck icps, iamp, kpick, iplk, idamp, ifilt, axcite

out apluck
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael A. Casey  
M.I.T.  
Cambridge, Mass.  
1997

Nouveau dans la Version 3.47

# wgpluck2

wgpluck2 — Modèle physique de corde pincée.

## Description

*wgpluck2* est une implémentation du modèle physique de corde pincée. On peut contrôler le point d'excitation, le point de lecture et le filtre. Basé sur l'algorithme de Karplus-Strong.

## Syntaxe

ares **wgpluck2** *iplk*, *kamp*, *icps*, *kpick*, *krefl*

## Initialisation

*iplk* -- Le point d'excitation est *iplk*, qui représente une fraction de la longueur de la corde (0 à 1). Un point d'excitation de zéro signifie l'absence d'excitation initiale.

*icps* -- La corde produit une hauteur de *icps*.

## Exécution

*kamp* -- Amplitude de la note.

*kpick* -- Fraction de la longueur de la corde où sera lue la sortie.

*krefl* -- le coefficient de réflexion, indiquant l'amortissement et le taux d'extinction. Il doit être strictement compris entre 0 et 1 (il n'acceptera pas 0 ni 1).

## Exemples

Voici un exemple de l'opcode *wgpluck2*. Il utilise le fichier *wgpluck2.csd* [examples/wgpluck2.csd].

### Exemple 606. Exemple de l'opcode *wgpluck2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d          ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o wgpluck2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  iplk = 0.75
  kamp = 30000
  icps = 220
  kpick = 0.75
  krefl = 0.5

  apluck wgpluck2 iplk, kamp, icps, kpick, krefl

  out apluck
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*repluck*

## Credits

Auteur : John ffitch (d'après Perry Cook)  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# wguide1

wguide1 — A simple waveguide model consisting of one delay-line and one first-order lowpass filter.

## Description

A simple waveguide model consisting of one delay-line and one first-order lowpass filter.

## Syntax

```
ares wguide1 asig, xfreq, kcutoff, kfeedback
```

## Performance

*asig* -- the input of excitation noise.

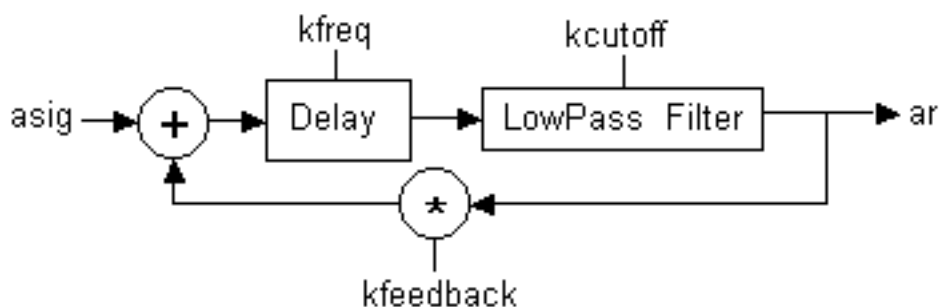
*xfreq* -- the frequency (i.e. the inverse of delay time) Changed to x-rate in Csound version 3.59.

*kcutoff* -- the filter cutoff frequency in Hz.

*kfeedback* -- the feedback factor.

*wguide1* is the most elemental waveguide model, consisting of one delay-line and one first-order low-pass filter.

Implementing waveguide algorithms as opcodes, instead of orc instruments, allows the user to set *kr* different than *sr*, allowing better performance particularly when using real-time.



wguide1.

## Examples

Here is an example of the wguide1 opcode. It uses the file *wguide1.csd* [examples/wguide1.csd].

### Exemple 607. Example of the wguide1 opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wguidel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a simple noise waveform.
instr 1
; Generate some noise.
asig noise 20000, 0.5

out asig
endin

; Instrument #2 - a waveguide example.
instr 2
; Generate some noise.
asig noise 20000, 0.5

; Run it through a wave-guide model.
kfreq init 200
kcutoff init 3000
kfeedback init 0.8
awg1 wguidel asig, kfreq, kcutoff, kfeedback

out awg1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*wguide2*

## Credits

Author: Gabriel Maldonado  
Italy  
October 1998

Example written by Kevin Conder.

New in Csound version 3.49



# wguide2

wguide2 — A model of beaten plate consisting of two parallel delay-lines and two first-order lowpass filters.

## Description

A model of beaten plate consisting of two parallel delay-lines and two first-order lowpass filters.

## Syntax

```
ares wguide2 asig, xfreq1, xfreq2, kcutoff1, kcutoff2, \  
      kfeedback1, kfeedback2
```

## Performance

*asig* -- the input of excitation noise

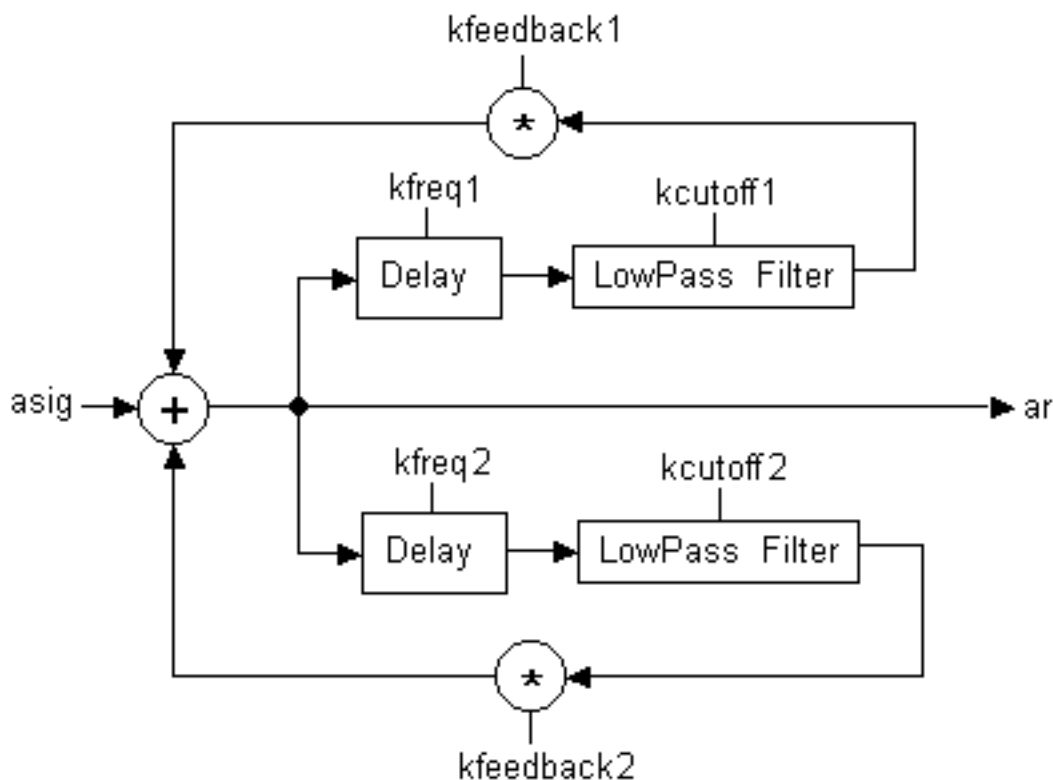
*xfreq1*, *xfreq2* -- the frequency (i.e. the inverse of delay time) Changed to x-rate in Csound version 3.59.

*kcutoff1*, *kcutoff2* -- the filter cutoff frequency in Hz.

*kfeedback1*, *kfeedback2* -- the feedback factor

*wguide2* is a model of beaten plate consisting of two parallel delay-lines and two first-order lowpass filters. The two feedback lines are mixed and sent to the delay again each cycle.

Implementing waveguide algorithms as opcodes, instead of orc instruments, allows the user to set *kr* different than *sr*, allowing better performance particularly when using real-time.



wguide2.



### Note

As a rule of thumb, to avoid making *wguide2* unstable, the sum of the two feedback values should be below 0.5.

## Examples

Here is an example of the *wguide2* opcode. It uses the file *wguide2.csd* [examples/wguide2.csd].

### Exemple 608. Example of the *wguide1* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wguide1.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
nchnls = 2
instr 1
```

```
afrq line 50, 10, 100
asig oscil 3000, afrq, 1
aenv expon 1,10,0.000001
aexc = aenv*asig
ares wguide2 aexc, 500, 1200, 777, 1500, 0.2, 0.25
      out ares,asig
endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1
i1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## See Also

*wguide1*

## Credits

Author: Gabriel Maldonado  
Italy  
October 1998

New in Csound version 3.49

Example written by John ffitch

# wiiconnect

wiiconnect — Lit des données provenant de l'un des contrôleurs Wiimote de Nintendo.

## Description

Ouvre et interroge au taux de contrôle de un à quatre contrôleurs externes Wiimote de Nintendo.

## Syntaxe

```
kres wiiconnect [itimeout, imaxnum]
```

## Initialisation

*itimeout* -- nombre entier de secondes pendant lesquelles le système doit attendre que toutes les Wiimotes soient connectées. S'il n'est pas spécifié, il vaut 10 secondes par défaut.

*imaxnum* -- nombre maximum de Wiimotes à repérer. S'il n'est pas spécifié, il vaut 4 par défaut.

Initialement, chaque Wiimote montre son allocation numérique en allumant une des quatre LEDs.

## Exécution

A chaque cycle de contrôle, chaque Wiimote est interrogée sur son état et sur sa position. Ces valeurs sont lues par l'opcode *wiidata*. Le résultat retourné vaut 1 la plupart du temps, mais sera nul si une Wiimote se déconnecte.

## Exemple

Voici un exemple des opcodes wii. Il utilise le fichier *wii.csd* [examples/wii.csd].

### Exemple 609. Exemple des opcodes wii.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-+rtaudio=alsa -o dac:hw:0
</CsOptions>
<CsInstruments>
nchnls = 2
ksmps = 400

#define WII_B          #3#
#define WII_A          #4#
#define WII_R_A        #304#
#define WII_PITCH      #20#
#define WII_ROLL       #21#
#define WII_BATTERY    #27#

#define WII_RUMBLE      #3#
#define WII_SET_LEDS   #4#

gkcnt init 1
```

```

instr 1
  il wiiconnect 3,1

  wiirange $WII_PITCH., -20, 0
  kb wiidata $WII_BATTERY.
  kt wiidata $WII_B.
  ka wiidata $WII_A.
  kra wiidata $WII_R_A.
  gka wiidata $WII_PITCH.
  gkp wiidata $WII_ROLL.
; If the B (trigger) button is pressed then activate a note
  if (kt==0) goto ee
  event "i", 2, 0, 5
  gkcnt = gkcnt + 1
  wiisend $WII_SET_LEDS., gkcnt
ee:
  if (ka==0) goto ff
  wiisend $WII_RUMBLE., 1
ff:
  if (kra==0) goto gg
  wiisend $WII_RUMBLE., 0
gg:
  printk2 kb
endin

instr 2
  a1 oscil ampdbs(gka), 440+gkp, 1
  outs a1, a1
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1
i1 0 300

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*wiidata, wiirange, wiisend*

## Crédits

Auteur : John ffitch  
 Codemist Ltd  
 2009

Nouveau dans la version 5.11

# wiidata

wiidata — Lit des données provenant de l'un des contrôleurs externes Wiimote de Nintendo.

## Description

Lit des données provenant de un à quatre contrôleurs externes Wiimote de Nintendo.

## Syntaxe

```
kres wiidata kcontrol[, knum]
```

## Initialisation

Cet opcode doit être utilisé de pair avec un opcode *wiiconnect* actif.

## Exécution

*kcontrol* -- le code du contrôle à lire

*knum* -- le numéro de la Wiimote à interroger, qui est par défaut la première.

A chaque accès, un type de donnée particulier de la Wiimote est lu. Les contrôles actuellement implémentés sont donnés ci-dessous, avec le nom de macro défini dans le fichier *wii\_mac* :

0 (WII\_BUTTONS) : retourne une combinaison de bits représentant tous les boutons enfoncés.

1 (WII\_TWO) : retourne 1 si le bouton vient d'être enfoncé, 0 sinon.

2 (WII\_ONE) : comme ci-dessus.

3 (WII\_B) : comme ci-dessus.

4 (WII\_A) : comme ci-dessus.

5 (WII\_MINUS) : comme ci-dessus.

8 (WII\_HOME) : comme ci-dessus.

9 (WII\_LEFT) : comme ci-dessus.

10 (WII\_RIGHT) : comme ci-dessus.

11 (WII\_DOWN) : comme ci-dessus.

12 (WII\_UP) : comme ci-dessus.

13 (WII\_PLUS) : comme ci-dessus.

Si le numéro du contrôle vaut 100 plus un de ces codes de bouton, l'état courant du bouton est retourné. Les macros telles que *WII\_S\_TWO*, etc sont définies pour cela.

Si le numéro du contrôle vaut 200 plus un de ces codes de bouton, la valeur retournée est 1 si le bouton est enfoncé, et 0 sinon. Les macros telles que *WII\_H\_TWO*, etc sont définies pour cela.

Si le numéro du contrôle vaut 300 plus un de ces codes de bouton, la valeur retournée est 1 si le bouton vient d'être relâché, et 0 sinon. Les macros telles que `WII_R_TWO`, etc sont définies pour cela.

20 (`WII_PITCH`) : L'inclinaison de la Wiimote. La valeur en degrés est comprise entre -90 et +90, à moins d'une modification de l'intervalle par un appel à *wiirange*.

21 (`WII_ROLL`) : La rotation de la Wiimote. La valeur en degrés est comprise entre -90 et +90, à moins d'une modification de l'intervalle par un appel à *wiirange*.

23 (`WII_FORCE_X`) : La force appliquée à la Wiimote selon les trois axes.

24 (`WII_FORCE_Y`) :

25 (`WII_FORCE_Z`) :

26 (`WII_FORCE_TOTAL`) : L'intensité totale de la force appliquée à la Wiimote.

27 (`WII_BATTERY`) : Le pourcentage de la charge des piles restante.

28 (`WII_NUNCHUK_ANG`) : L'angle du joystick du nunchuk en degrés.

29 (`WII_NUNCHUK_MAG`) : Le déplacement du joystick du nunchuk par rapport à sa position centrale.

30 (`WII_NUNCHUK_PITCH`) : L'inclinaison du nunchuk en degrés, comprise entre -90 et +90, à moins d'une modification de l'intervalle par un appel à *wiirange*.

31 (`WII_NUNCHUK_ROLL`) : La rotation du nunchuk en degrés, comprise entre -90 et +90, à moins d'une modification de l'intervalle par un appel à *wiirange*.

33 (`WII_NUNCHUK_Z`): L'état du bouton Z du nunchuk.

34 (`WII_NUNCHUK_C`): L'état du bouton C du nunchuk.

35 (`WII_IR1_X`): Le pointage infrarouge de la Wiimote.

36 (`WII_IR1_Y`):

37 (`WII_IR1_Z`):

## Exemples

Voir l'exemple de *wiiconnect*.

## Voir Aussi

*wiiconnect*, *wiirange*, *wiisend*,

## Crédits

Auteur : John ffitich  
Codemist Ltd  
2009

Nouveau dans la version 5.11

# wiirange

wiirange — Fixe l'échelle et les limites de l'intervalle de certains des paramètres de la Wiimote.

## Description

Fixe l'échelle et les limites de l'intervalle de certains des paramètres de la Wiimote.

## Syntaxe

```
wiirange icontrol, iminimum, imaximum[, inum]
```

## Initialisation

Cet opcode doit être utilisé de pair avec un opcode *wiiconnect* actif.

*icontrol* -- numéro du contrôle à pondérer. C'est l'un des suivants : 20 (WII\_PITCH), 21 (WII\_ROLL), 30 (WII\_NUNCHUK\_PITCH), 31 (WII\_NUNCHUK\_ROLL).

*iminimum* -- valeur minimale du contrôle.

*imaximum* -- valeur maximale du contrôle.

## Exemples

Voir l'exemple de *wiiconnect*.

## Voir Aussi

*wiiconnect*, *wiidata*, *wiisend*,

## Crédits

Auteur : John ffitich  
Codemist Ltd  
2009

Nouveau dans la version 5.11



# wiisend

wiisend — Envoie des données à l'un des contrôleurs externes Wiimote de Nintendo.

## Description

Envoie des données à l'un des contrôleurs externes Wiimote de Nintendo.

## Syntaxe

```
kres wiisend kcontrol, kvalue[, knum]
```

## Initialisation

Cet opcode doit être utilisé de pair avec un opcode *wiiconnect* actif.

## Exécution

*kcontrol* -- le code du contrôle à écrire.

*kvalue* -- la valeur à écrire dans le contrôle.

*knum* -- le numéro de la Wiimote de destination, qui est par défaut la première (zéro).

A chaque accès, un élément de donnée particulier de la Wiimote est écrit. Les contrôles actuellement implémentés sont donnés ci-dessous, avec le nom de macro défini dans le fichier *wii\_mac* :

3 (WII\_RUMBLE) : démarre ou arrête le vibreur de la Wiimote, selon la valeur de *kvalue* (0 pour arrêter, 1 pour démarrer).

4 (WII\_SET\_LEDS) : positionne les quatre LEDs de la Wiimote selon la représentation binaire de *kvalue*.

## Exemples

Voir l'exemple de *wiiconnect*.

## Voir Aussi

*wiiconnect*, *wiidata*, *wiirange*,

## Crédits

Auteur : John ffitich  
Codemist Ltd  
2009

Nouveau dans la version 5.11

# wrap

`wrap` — Wraps-around the signal that exceeds the low and high thresholds.

## Description

Wraps-around the signal that exceeds the low and high thresholds.

## Syntax

```
ares wrap asig, klow, khigh
```

```
ires wrap isig, ilow, ihigh
```

```
kres wrap ksig, klow, khigh
```

## Initialization

*isig* -- input signal

*ilow* -- low threshold

*ihigh* -- high threshold

## Performance

*xsig* -- input signal

*klow* -- low threshold

*khigh* -- high threshold

*wrap* wraps-around the signal that exceeds the low and high thresholds.

This opcode is useful in several situations, such as table indexing or for clipping and modeling a-rate, i-rate or k-rate signals. *wrap* is also useful for wrap-around of table data when the maximum index is not a power of two (see *table* and *tablei*). Another use of *wrap* is in cyclical event repeating, with arbitrary cycle length.

## See Also

*limit*, *mirror*

## Credits

Author: Gabriel Maldonado  
Italy

New in Csound version 3.49

# wterrain

wterrain — Un opcode simple de synthèse par terrain d'onde.

## Description

Un opcode simple de synthèse par terrain d'onde.

## Syntaxe

```
aout wterrain kamp, kpch, k_xcenter, k_ycenter, k_xradius, k_yradius, \  
      itabx, itaby
```

## Initialisation

*itabx, itaby* -- Les deux tables qui définissent le terrain.

## Exécution

La sortie est le résultat du dessin d'une ellipse dont les axes *k\_xradius* et *k\_yradius* centrés en (*k\_xcenter*, *k\_ycenter*), et de sa traversée à la fréquence *kpch*.

## Exemples

Voici un exemple de l'opcode wterrain. Il utilise le fichier *wterrain.csd* [examples/wterrain.csd].

### Exemple 610. Exemple de l'opcode wterrain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out   Audio in   No messages  
-odac        -iadc       -d      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o wterrain.wav -W ;;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
; Initialize the global variables.  
sr = 44100  
kr = 4410  
ksmps = 10  
nchnls = 1  
  
instr 1  
kdclk linseg 0, 0.01, 1, p3-0.02, 1, 0.01, 0  
kcx line 0.1, p3, 1.9  
krx linseg 0.1, p3/2, 0.5, p3/2, 0.1  
kpch line cpspch(p4), p3, p5 * cpspch(p4)  
a1 wterrain 10000, kpch, kcx, kcx, -krx, krx, p6, p7  
a1 dcblock a1  
      out a1*kdclk  
endin
```

```
</CsInstruments>
<CsScore>

f1      0      8192    10      1 0 0.33 0 0.2 0 0.14 0 0.11
f2      0      4096    10      1

i1      0      4      7.00 1 1 1
i1      4      4      6.07 1 1 2
i1      8      8      6.00 1 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Matthew Gillard  
Nouveau dans la version 4.19

## xadsr

`xadsr` — Calcule l'enveloppe ADSR classique.

## Description

Calcule l'enveloppe ADSR classique.

## Syntaxe

```
ares xadsr iatt, idec, islev, irel [, idel]
```

```
kres xadsr iatt, idec, islev, irel [, idel]
```

## Initialisation

*iatt* -- durée de l'attaque (attack)

*idec* -- durée de la première chute (decay)

*islev* -- niveau d'entretien (sustain)

*irel* -- durée de la chute (release)

*idel* -- délai de niveau zéro avant le démarrage de l'enveloppe

## Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

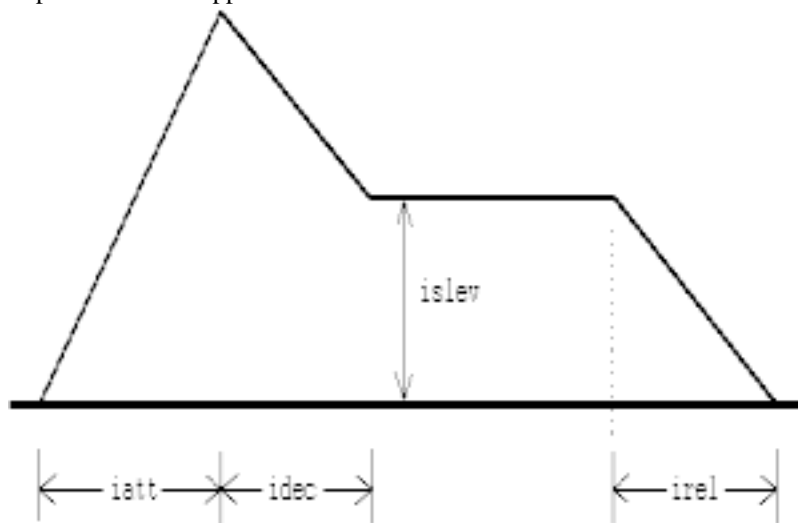


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *xadsr* n'est pas adapté au traitement des événements MIDI, pour lesquels il faut plutôt utiliser *mxadsr*.

L'opcode *xadsr* est identique à *adsr* sauf qu'il utilise des segments exponentiels plutôt que linéaires.

*xadsr* est nouveau dans la version 3.51 de Csound.

## Voir Aussi

*adsr, madsr, mxadsr*

## Crédits

Auteur : John ffitich

# xin

xin — Passe des variables à un bloc d'opcode défini par l'utilisateur.

## Description

Les opcodes *xin* et *xout* copient des variables vers et depuis la définition de l'opcode, permettant la communication avec l'instrument appelant.

Les types des variables d'entrée et de sortie sont définis par les paramètres *intypes* et *outtypes*.



### Notes

- *xin* et *xout* ne doivent être appelés qu'une fois, et *xin* doit précéder *xout*, sinon il pourra y avoir une erreur d'initialisation et une désactivation de l'instrument courant.
- Ces opcodes ne sont exécutés que pendant l'initialisation. La copie pendant l'exécution se fait par l'appel de l'opcode défini par l'utilisateur. Cela veut dire que si l'on veut ignorer *xin* ou *xout* avec *kgoto*, cela ne marche pas alors que *igoto* affecte à la fois les opérations de l'initialisation et de l'exécution.

## Syntaxe

```
xinarg1 [, xinarg2] ... [xinargN] xin
```

## Exécution

*xinarg1*, *xinarg2*, ... - arguments d'entrée. Le nombre et le type des variables doit concorder avec la déclaration *intypes* de l'opcode défini par l'utilisateur. Cependant *xin* ne vérifie pas si l'utilisation des variables d'initialisation et du taux de contrôle est correcte.

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

On peut alors utiliser le nouvel opcode avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN] name [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

## Exemples

Voir l'exemple de l'opcode *opcode*.

## Voir Aussi

*endop, opcode, setksmps, xout*

## Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code par Matt J. Ingalls

Nouveau dans la version 4.22



# xout

**xout** — Récupère les variables d'un bloc d'opcode défini par l'utilisateur.

## Description

Les opcodes *xin* et *xout* copient des variables vers et depuis la définition de l'opcode, permettant la communication avec l'instrument appelant.

Les types des variables d'entrée et de sortie sont définis par les paramètres *intypes* et *outtypes*.



### Notes

- *xin* et *xout* ne doivent être appelés qu'une fois, et *xin* doit précéder *xout*, sinon il pourra y avoir une erreur d'initialisation et une désactivation de l'instrument courant.
- Ces opcodes ne sont exécutés que pendant l'initialisation. La copie pendant l'exécution se fait par l'appel de l'opcode défini par l'utilisateur. Cela veut dire que si l'on veut ignorer *xin* ou *xout* avec *kgoto*, cela ne marche pas alors que *igoto* affecte à la fois les opérations de l'initialisation et de l'exécution.

## Syntaxe

```
xout xoutarg1 [, xoutarg2] ... [, xoutargN]
```

## Exécution

*xoutarg1*, *xoutarg2*, ... - arguments de sortie. Le nombre et le type des variables doit concorder avec la déclaration *outtypes* de l'opcode défini par l'utilisateur. Cependant *xout* ne vérifie pas si l'utilisation des variables d'initialisation et du taux de contrôle est correcte.

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

On peut alors utiliser le nouvel opcode avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN] name [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

## Exemples

Voir l'exemple de l'opcode *opcode*.

## Voir Aussi

*endop, opcode, setksmps, xin*

## Crédits

Auteur : Istvan Varga, 2002; basé sur du code par Matt J. Ingalls

Nouveau dans la version 4.22

# xscanmap

xscanmap — Permet de lire la position et la vitesse d'un noeud dans une procédure de balayage.

## Description

Permet de lire la position et la vitesse d'un noeud dans une procédure de balayage.

## Syntaxe

```
kpos, kvel xscanmap iscan, kamp, kvamp [, iwhich]
```

## Initialisation

*iscan* -- la procédure de balayage à lire

*iwhich* (facultatif) -- le noeud à tester. 0 par défaut.

## Exécution

*kamp* -- facteur d'amplification de la valeur *kpos*.

*kvamp* -- facteur d'amplification de la valeur *kvel*.

L'état interne d'un noeud est lu. Cela comprend sa position et sa vitesse. Ils sont amplifiés par les valeurs *kamp* et *kvamp*.

## Crédits

Auteur : John ffitich

Nouveau dans la version 4.20

# xscansmap

xscansmap — Permet de lire la position et la vitesse d'un noeud dans une procédure de balayage.

## Description

Permet de lire la position et la vitesse d'un noeud dans une procédure de balayage.

## Syntaxe

```
xscansmap kpos, kvel, iscan, kamp, kvamp [, iwhich]
```

## Initialisation

*iscan* -- la procédure de balayage à lire

*iwhich* (facultatif) -- le noeud à tester. 0 par défaut.

## Exécution

*kpos* -- la position du noeud.

*kvel* -- la vitesse du noeud.

*kamp* -- facteur d'amplification de la valeur *kpos*.

*kvamp* -- facteur d'amplification de la valeur *kvel*.

L'état interne d'un noeud est lu. Cela comprend sa position et sa vitesse. Ils sont amplifiés par les valeurs *kamp* et *kvamp*.

## Crédits

Nouveau dans la version 4.21

Novembre 2002. Merci à Rasmus Ekman pour avoir précisé cet opcode.

## xscans

xscans — Générateur rapide de forme d'onde et de la table d'onde de la synthèse par balayage.

## Description

Version expérimentale de *scans*. Autorise des matrices bien plus grandes, est plus rapide et plus compact, mais supprime une certaine flexibilité (non utilisée ?). S'il est apprécié, il remplacera l'ancien opcode car sa syntaxe est compatible bien qu'étendue.

## Syntaxe

```
ares xscans kamp, kfreq, ifntraj, id [, iorder]
```

## Initialisation

*ifntraj* -- table contenant la trajectoire du balayage. C'est une série de nombres qui contiennent les adresses des masses. L'ordre de ces adresses est utilisé comme chemin de balayage. Ne doit pas contenir de valeurs supérieures au nombre de masses, ou des nombres négatifs. Voir l'*introduction à la section sur la synthèse par balayage*.

*id* -- s'il est positif, c'est l'ID de l'opcode. Il est utilisé pour relier l'opcode de balayage au bon générateur de forme d'onde. S'il est négatif, sa valeur absolue indique la table d'onde dans laquelle sera écrite la forme d'onde. Cette forme d'onde peut être utilisée par la suite par un autre opcode pour générer du son. Le contenu initial de cette table sera écrasé.

*iorder* (facultatif, 0 par défaut) -- ordre de l'interpolation utilisée en interne. Peut prendre n'importe quelle valeur comprise entre 1 et 4, et vaut 4 par défaut, qui est l'interpolation quartique. 2 est l'interpolation quadratique et 1 l'interpolation linéaire. Les nombres les plus élevés donnent un traitement plus lent, mais pas nécessairement meilleur.

## Exécution

*kamp* -- amplitude de la sortie. Noter que l'amplitude résultante dépend aussi des valeurs instantanées de la table d'onde. Ce nombre est en fait la facteur de pondération de la table d'onde.

*kfreq* -- fréquence de balayage

## Format de Matrice

Le nouveau format de matrice est une liste de connexions, une par ligne reliant le point x au point y. Aucun poids n'est affecté au lien ; il est supposé valoir l'unité. La liste est précédée par la ligne <MATRIX> et se termine par une ligne </MATRIX>

Par exemple, une corde circulaire de 8 sera codée par

```
<MATRIX>
0 1
1 0
1 2
2 1
2 3
3 2
</MATRIX>
```

```
3 4
4 3
4 5
5 4
5 6
6 5
6 7
7 6
0 7
</MATRIX>
```

## Crédits

Ecrit par John ffitch.

Nouveau dans la version 4.20

## Exemples

Pour un exemple, voir la documentation de *scans*.

## Voir Aussi

*scans*, *xscanu*

# xscanu

xscanu — Calcule la forme d'onde et la table d'onde à utiliser dans la synthèse par balayage.

## Description

Version expérimentale de *scanu*. Autorise des matrices bien plus grandes, est plus rapide et plus compact, mais supprime une certaine flexibilité (non utilisée ?). S'il est apprécié, il remplacera l'ancien opcode car sa syntaxe est compatible bien qu'étendue.

## Syntaxe

```
xscanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
      kstif, kcentr, kdamp, ileft,  iright, kpos, kstrngth, ain, idisp, id
```

## Initialisation

*init* -- la position initiale des masses. Si c'est un nombre négatif, alors la valeur absolue de *init* indique la table à utiliser pour la forme du marteau. Si *init* > 0, il doit représenter le nombre de masses attendu, sinon sa valeur est sans importance.

*irate* -- taux de mise à jour.

*ifnvel* -- ftable contenant la vitesse initiale de chaque masse. Sa taille est le nombre de masses attendu.

*ifnmass* -- ftable contenant la valeur de chaque masse. Sa taille est le nombre de masses attendu.

*ifnstif* --

- soit une ftable contenant la raideur du ressort de chaque connexion. Sa taille est le carré du nombre de masses attendu. Ses données sont ordonnées selon la succession des lignes de la matrice de connexion du système.
- soit une chaîne de caractères donnant le nom d'un fichier au format MATRIX

*ifncentr* -- ftable contenant la force de centrage de chaque masse. Sa taille est le nombre de masses attendu.

*ifndamp* -- ftable contenant le facteur d'amortissement de chaque masse. Sa taille est le nombre de masses attendu.

*ileft* -- si *init* < 0, position du marteau de gauche (*ileft* = 0 frappe complètement à gauche, *ileft* = 1 frappe complètement à droite).

*iright* -- si *init* < 0, position du marteau de droite (*iright* = 0 frappe complètement à gauche, *iright* = 1 frappe complètement à droite).

*idisp* -- s'il vaut 0, il n'y a pas d'affichage des masses.

*id* -- s'il est positif, c'est l'ID de l'opcode. Il est utilisé pour relier l'opcode de balayage au bon générateur de forme d'onde. S'il est négatif, sa valeur absolue indique la table d'onde dans laquelle sera écrite la forme d'onde. Cette forme d'onde peut être utilisée par la suite par un autre opcode pour générer du son.

Le contenu initial de cette table sera écrasé.

## Exécution

*kmass* -- pondère les masses

*kstif* -- pondère la raideur des ressorts

*kcentr* -- pondère la force de centrage

*kdamp* -- pondère l'amortissement

*kpos* -- position d'un marteau actif le long de la corde (*kpos* = 0 est complètement à gauche, *kpos* = 1 est complètement à droite). La forme du marteau est déterminée par *init* et sa puissance de percussion est *kstrngth*.

*kstrngth* -- puissance utilisée par le marteau actif

*ain* -- entrée audio qui s'ajoute à la vitesse des masses. L'amplitude ne doit pas être trop grande.

## Format de Matrice

Le nouveau format de matrice est une liste de connexions, une par ligne reliant le point x au point y. Aucun poids n'est affecté au lien ; il est supposé valoir l'unité. La liste est précédée par la ligne <MATRIX> et se termine par une ligne </MATRIX>

Par exemple, une corde circulaire de 8 sera codée par

```
<MATRIX>
0 1
1 0
1 2
2 1
2 3
3 2
3 4
4 3
4 5
5 4
5 6
6 5
6 7
7 6
0 7
</MATRIX>
```

## Crédits

Ecrit par John ffitich.

Nouveau dans la version 4.20

## Exemples

Pour un exemple, voir la documentation de *scans*.

## Voir Aussi



*scanu, xscans*

# xtratism

xtratism — Extend the duration of real-time generated events.

## Description

Extend the duration of real-time generated events and handle their extra life (Usually for usage along with *release* instead of *linenr*, *linsegr*, etc).

## Syntax

```
xtratism iextradur
```

## Initialization

*iextradur* -- additional duration of current instrument instance

## Performance

*xtratism* extends current MIDI-activated note duration by *iextradur* seconds after the corresponding noteoff message has deactivated the current note itself. It is usually used in conjunction with *release*. This opcode has no output arguments.

This opcode is useful for implementing complex release-oriented envelopes, whose duration is not known when the envelope starts (e.g. for real-time MIDI generated events).

## Examples

Here is a simple example of the xtratism opcode. It uses the file *xtratism.csd* [examples/xtratism.csd].

### Exemple 611. Example of the xtratism opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

This example shows how to generate a release segment for an ADSR envelope after a MIDI noteoff is received, extending the duration with *xtratism* and using *release* to check whether the note is on the release phase.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent  MIDI in
-odac        -iadc      -d      -M0    ;;realtime I/O
</CsOptions>
<CsInstruments>
;Simple usage of the xtratism opcode

instr 1
    inum notnum
    icps cpsmidi
    iamp  ampmidi 4000
;
```

```

;----- complex envelope block -----
xtratism 1 ;extra-time, i.e. release dur
krel init 0
krel release ;outputs release-stage flag (0 or 1 values)
if (krel == 1) kgoto rel ;if in release-stage goto release section
;
;***** attack and sustain section *****
kmp1 linseg 0, .03, 1, .05, 1, .07, 0, .08, .5, 4, 1, 50, 1
kmp = kmp1*iamp
kgoto done
;
;----- release section -----
rel:
kmp2 linseg 1, .3, .2, .7, 0
kmp = kmp1*kmp2*iamp
done:
;-----
al oscili kmp, icps, 1
out al
endin

</CsInstruments>
<CsScore>
f 0 3600 ;dummy table to wait for realtime MIDI events
e
</CsScore>
</CsoundSynthesizer>

```

Here is a more elaborate example of the xtratism opcode. It uses the file *xtratism-2.csd* [examples/xtratism-2.csd].

## Exemple 612. More complex example of the xtratism opcode.

This example shows how to generate a release segment for an ADSR envelope after a MIDI noteoff is received, extending the duration with *xtratism* and using *release* to check whether the note is on the release phase. Two envelopes are generated simultaneously for the left and right channels.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent  MIDI in
-odac        -iadc     -d      -M0    ;;realtime I/O
</CsOptions>
<CsInstruments>
;xtratism example by Jonathan Murphy Dec. 2006

sr          = 44100
ksmps       = 32
nchnls      = 2

; sine wave for oscillators
gisin       ftgen 1, 0, 4096, 10, 1
; set volume initially to midpoint
ctrlinit 1, 7, 64

;;; simple two oscil, two envelope synth
instr 1

; frequency
kcps        cpsmidib
; initial velocity (noteon)
ivel        veloc

; master volume
kamp        ctrl7 1, 7, 0, 127
kamp        = kamp * ivel

; parameters for aenv1
iatt1       = 0.03
idecl       = 1
isus1       = 0.25
irell       = 1

```

```

; parameters for aenv2
iatt2    = 0.06
idec2    = 2
isus2    = 0.5
irel2    = 2

; extra (release) time allocated
xtratism (irel1>irel2 ? irel1 : irel2)
; krel is used to trigger envelope release
krel     init 0
krel     release
; if noteoff received, krel == 1, otherwise krel == 0
if (krel == 1) kgoto rel

; attack, decay, sustain segments
atmp1    linseg 0, iatt1, 1, idec1, isus1 , 1, isus1
atmp2    linseg 0, iatt2, 1, idec2, isus2 , 1, isus2
aenv1    = atmp1
aenv2    = atmp2
kgoto    done

; release segment
rel:
atmp3    linseg 1, irel1, 0, 1, 0
atmp4    linseg 1, irel2, 0, 1, 0
aenv1    = atmp1 * atmp3 ;to go from the current value (in case
aenv2    = atmp2 * atmp4 ;the attack hasn't finished) to the release.

; control oscillator amplitude using envelopes
done:
aoscl    oscil aenv1, kcps, 1
aoscl    oscil aenv2, kcps * 1.5, 1
aoscl    = aoscl * kamp
aoscl    = aoscl * kamp

; send aoscl to left channel, aoscl2 to right,
; release times are noticeably different
outs     aoscl, aoscl2

endin

</CsInstruments>
<CsScore>

f 0 3600 ;dummy table to wait for realtime MIDI events

</CsScore>
</CsoundSynthesizer>

```

## See Also

*linenr, release*

## Credits

Author: Gabriel Maldonado

Italy

Examples by Gabriel Maldonado and Jonathan Murphy

New in Csound version 3.47

# xyin

xyin — Sense the cursor position in an output window

## Description

Sense the cursor position in an output window. When *xyin* is called the position of the mouse within the output window is used to reply to the request. This simple mechanism does mean that only one *xyin* can be used accurately at once. The position of the mouse is reported in the output window.

## Syntax

```
kx, ky xyin iprd, ixmin, ixmax, iymn, iymax [, ixinit] [, iyinit]
```

## Initialization

*iprd* -- period of cursor sensing (in seconds). Typically .1 seconds.

*xmin*, *xmax*, *ymin*, *ymax* -- edge values for the x-y coordinates of a cursor in the input window.

*ixinit*, *iyinit* (optional) -- initial x-y coordinates reported; the default values are 0,0. If these values are not within the given min-max range, they will be coerced into that range.

## Performance

*xyin* samples the cursor x-y position in an input window every *iprd* seconds. Output values are repeated (not interpolated) at the k-rate, and remain fixed until a new change is registered in the window. There may be any number of input windows. This unit is useful for real-time control, but continuous motion should be avoided if *iprd* is unusually small.



### Note

Depending on your platform and distribution, you might need to enable displays using the *-displays* command line flag.

## Examples

Here is an example of the *xyin* opcode. It uses the file *xyin.csd* [examples/xyin.csd].

### Exemple 613. Example of the *xyin* opcode.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc     --displays ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```

; -o xyin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print and capture values every 0.1 seconds.
iprd = 0.1
; The x values are from 1 to 30.
ixmin = 1
ixmax = 30
; The y values are from 1 to 30.
iymin = 1
iymax = 30
; The initial values for X and Y are both 15.
ixinit = 15
iyinit = 15

; Get the values kx and ky using the xyin opcode.
kx, ky xyin iprd, ixmin, ixmax, iymin, iymax, ixinit, iyinit

; Print out the values of kx and ky.
printks "kx=%f, ky=%f\n", iprd, kx, ky

; Play an oscillator, use the x values for amplitude and
; the y values for frequency.
kamp = kx * 1000
kcps = ky * 220
a1 oscil kamp, kcps, 1

out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 30 seconds.
i 1 0 30
e

</CsScore>
</CsoundSynthesizer>

```

As the values of kx and ky change, they will be printed out like this:

```

kx=8.612036, ky=22.677933
kx=10.765685, ky=15.644135

```

## Credits

Example written by Kevin Conder.

# zaci

zaci — Efface une ou plusieurs variables dans l'espace za.

## Description

Efface une ou plusieurs variables dans l'espace za.

## Syntaxe

```
zaci kfirst, klast
```

## Exécution

*kfirst* -- Première position zk ou za de l'intervalle à effacer.

*klast* -- Dernière position zk ou za de l'intervalle à effacer.

*zaci* efface une ou plusieurs variables dans l'espace za. Ceci est utile pour les variables utilisées comme accumulateur pour mélanger des signaux de taux-a à chaque cycle, mais qui doivent être effacés avant le prochain groupe de calculs.

## Exemples

Voici un exemple de l'opcode zaci. Il utilise le fichier *zaci.csd* [examples/zaci.csd].

### Exemple 614. Exemple de l'opcode zaci.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o zaci.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Generate a simple sine waveform.
asin oscil 20000, 440, 1

; Send the sine waveform to za variable #1.
zaw asin, 1
```

```
endin

; Instrument #2 -- generates audio output.
instr 2
; Read za variable #1.
al zar 1

; Generate the audio output.
out al

; Clear the za variables, get them ready for
; another pass.
zacr 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zamod, zar, zaw, zawm, ziw, ziwm*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.



# zakinit

zakinit — Etablit l'espace zak.

## Description

Etablit l'espace zak. Ne doit être appelé qu'une seule fois.

## Syntaxe

```
zakinit isizea, isizek
```

## Initialisation

*isizea* -- le nombre de positions de taux audio pour les patch de taux-a. Chaque position est un tableau de longueur *ksmps*.

*isizek* -- le nombre de positions à réserver pour les nombres en virgule flottante dans l'espace zk. On peut lire et écrire dans celles-ci au taux-i et au taux-k.

## Exécution

Il y a au moins une position d'allouée pour chaque espace za et zk. Il peut y avoir des milliers ou des dizaines de milliers de positions za et zk, mais la plupart des pièces n'en nécessitent probablement que quelques douzaines pour patcher les signaux. Ces positions de patch sont référencées par un numéro dans les autres opcodes zak.

Pour n'exécuter *zakinit* qu'une seule fois, on le place en dehors de toute définition d'instrument, dans l'en-tête de l'orchestre, après *sr*, *kr*, *ksmps*, et *nchnls*.



### Note

Les canaux zak se comptent à partir de 0, si bien que si l'on définit un canal, le seul canal valide est le canal 0.

## Exemples

Voici un exemple de l'opcode *zakinit*. Il utilise le fichier *zakinit.csd* [examples/zakinit.csd].

### Exemple 615. Exemple de l'opcode *zakinit*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```

; -o zakinit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 4410
nchnls = 1

; Initialize the ZAK space.
; Create 3 a-rate variables and 5 k-rate variables.
zakinit 2, 3

instr 1 ;a simple waveform.
; Generate a simple sine waveform.
asin oscil 20000, 440, 1

; Send the sine waveform to za variable #1.
zaw asin, 1
endin

instr 2 ;generates audio output.
; Read za variable #1.
a1 zar 1

; Generate audio output.
out a1

; Clear the za variables, get them ready for
; another pass.
zacr 0, 3
endin

instr 3 ;increments k-type channels
k0 zkr 0
k1 zkr 1
k2 zkr 2

zkw k0+1, 0
zkw k1+5, 1
zkw k2+10, 2
endin

instr 4 ;displays values from k-type channels
k0 zkr 0
k1 zkr 1
k2 zkr 2

; The total count for k0 is 30, since there are 10
; control blocks per second and instruments 3 and 4
; are on for 3 seconds.
printf "k0 = %i\n", k0, k0
printf "k1 = %i\n", k1, k1
printf "k2 = %i\n", k2, k2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

i 1 0 1
i 2 0 1

i 3 0 3
i 4 0 3
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Robin Whittle  
Australie

Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zamod

zamod — Module un signal de taux-a par un autre.

## Description

Module un signal de taux-a par un autre.

## Syntaxe

```
ares zamod asig, kzamod
```

## Exécution

*asig* -- Le signal d'entrée

*kzamod* -- Contrôle quelle variable *za* sera utilisée pour la modulation. Une valeur positive indique une modulation additive, une valeur négative indique une modulation multiplicative. Une valeur de 0 ne fait aucun changement à *asig*.

*zamod* Module un signal de taux-a par un autre, qui provient d'une variable *za*. La position de la variable modulante est contrôlée par la variable de taux-i ou de taux-k *kzamod*. Ceci est la version de taux-a de *zkmod*.

## Exemples

Voici un exemple de l'opcode *zamod*. Il utilise le fichier *zamod.csd* [examples/zamod.csd].

### Exemple 616. Exemple de l'opcode *zamod*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zamod.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 2 a-rate variables and 2 k-rate variables.
zakinit 2, 2

; Instrument #1 -- a simple waveform.
instr 1
; Vary an a-rate signal linearly from 20,000 to 0.
asig line 20000, p3, 0
```

```
; Send the signal to za variable #1.
zaw asig, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Generate a simple sine wave.
asin oscil 1, 440, 1

; Modify the sine wave, multiply its amplitude by
; za variable #1.
a1 zamod asin, -1

; Generate the audio output.
out a1

; Clear the za variables, prepare them for
; another pass.
zacl 0, 2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zaci, ziw, ziwm*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zar

zir — Lecture à partir d'une position dans l'espace za au taux-a.

## Description

Lecture à partir d'une position dans l'espace za au taux-a.

## Syntaxe

ares **zar** kndx

## Exécution

*kndx* -- pointe sur la position za à lire.

*zar* lit la suite de nombres décimaux à *kndx* dans l'espace za, qui sont les ksmps nombres décimaux de taux-a à traiter dans un cycle-k.

## Exemples

Voici un exemple du opcode zar. Il utilise le *zar.csd* [exemples/zar.csd].

### Exemple 617. Exemple de l'opcode zar.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac       -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zar.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Generate a simple sine waveform.
asin oscil 20000, 440, 1

; Send the sine waveform to za variable #1.
zaw asin, 1
endin

; Instrument #2 -- generates audio output.
instr 2
```

```
; Read za variable #1.
a1 zar 1

; Generate audio output.
out a1

; Clear the za variables, get them ready for
; another pass.
zaci 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zarg, zir, zkr*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zarg

zarg — Lecture à partir d'une position dans l'espace za au taux-a avec application d'un gain.

## Description

Lecture à partir d'une position dans l'espace za au taux-a avec application d'un gain.

## Syntaxe

```
ares zarg kndx, kgain
```

## Initialisation

*kndx* -- pointe sur la position za à lire.

*kgain* -- Multiplicateur pour le signal taux-a.

## Exécution

*zarg* lit la suite de nombres décimaux à *kndx* dans l'espace za, qui sont les *ksmps* nombres décimaux de taux-a à traiter dans un cycle-k. *zarg* multiplie aussi le signal de taux-a par la valeur de taux-k *kgain*.

## Exemples

Voici un exemple de l'opcode *zarg*. Il utilise le fichier *zarg.csd* [examples/zarg.csd].

### Exemple 618. Exemple de l'opcode *zarg*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zarg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Generate a simple sine waveform, with an amplitude
; between 0 and 1.
asin oscil 1, 440, 1
```



```
; Send the sine waveform to za variable #1.
zaw asin, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Read za variable #1, multiply its amplitude by 20,000.
a1 zarg 1, 20000

; Generate audio output.
out a1

; Clear the za variables, get them ready for
; another pass.
zaci 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zar, zir, zkr*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zaw

zaw — Ecrit dans une variable za au taux-a sans mixage.

## Description

Ecrit dans une variable za au taux-a sans mixage.

## Syntaxe

**zaw** *asig*, *kndx*

## Exécution

*asig* -- Valeur à écrire dans la position za.

*kndx* -- Pointe sur la position zk ou za vers laquelle écrire.

*zaw* écrit *asig* dans la variable za spécifiée par *kndx*.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces zk ou za. Sinon, une erreur est rapportée, la valeur 0 est retournée, et il n'y a aucune écriture.

## Exemples

Voici un exemple de l'opcode zaw. Il utilise le fichier *zaw.csd* [examples/zaw.csd].

### Exemple 619. Exemple de l'opcode zaw.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc          -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zaw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Generate a simple sine waveform.
asin oscil 20000, 440, 1

; Send the sine waveform to za variable #1.
```

```
    zaw asin, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Read za variable #1.
a1 zar 1

; Generate the audio output.
out a1

; Clear the za variables, get them ready for
; another pass.
zACL 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zawm, ziw, ziwm, zkw, zkwm*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

## zawm

zawm — Ecrit dans une variable za au taux-a avec mixage.

## Description

Ecrit dans une variable za au taux-a avec mixage.

## Syntaxe

```
zawm asig, kndx [, imix]
```

## Initialisation

*imix* (facultatif, par défaut=1) -- indique si le mixage sera fait.

## Exécution

*asig* -- valeur à écrire dans l'espace za.

*kndx* -- Pointe sur la position zk ou za vers laquelle écrire.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces zk ou za. Sinon, une erreur est rapportée, la valeur 0 est retournée et il n'y a aucune écriture.

*zawm* est un opcode de mixage, il ajoute le signal à la valeur actuelle de la variable. Si aucun *imix* n'est spécifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw*, et *zaw*. Toute autre valeur entraînera un mixage.

*Avertissement* : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm*, et *zawm*, il faut faire attention à ce que les variables qui reçoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche possible serait d'établir certains intervalles de variables zk ou za à utiliser pour le mixage, puis d'utiliser ensuite *zkcl* ou *zawl* pour effacer ces variables.

## Exemples

Voici un exemple de l'opcode *zawm*. Il utilise le fichier *zawm.csd* [examples/zawm.csd].

### Exemple 620. Exemple de l'opcode *zawm*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o zawm.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a basic instrument.
instr 1
; Generate a simple sine waveform.
asin oscil 15000, 440, 1

; Mix the sine waveform with za variable #1.
zawm asin, 1
endin

; Instrument #2 -- another basic instrument.
instr 2
; Generate another waveform with a different frequency.
asin oscil 15000, 880, 1

; Mix this sine waveform with za variable #1.
zawm asin, 1
endin

; Instrument #3 -- generates audio output.
instr 3
; Read za variable #1, containing both waveforms.
a1 zar 1

; Generate the audio output.
out a1

; Clear the za variables, get them ready for
; another pass.
zacr 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
; Play Instrument #3 for one second.
i 3 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zaw, ziw, ziwm, zkw, zkwm*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

## zfilter2

`zfilter2` — Performs filtering using a transposed form-II digital filter lattice with radial pole-shearing and angular pole-warping.

### Description

General purpose custom filter with time-varying pole control. The filter coefficients implement the following difference equation:

$$(1)*y(n) = b0*x[n] + b1*x[n-1] + \dots + bM*x[n-M] - a1*y[n-1] - \dots - aN*y[n-N]$$

the system function for which is represented by:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + \dots + bM*Z^{-M}}{1 + a1*Z^{-1} + \dots + aN*Z^{-N}}$$

### Syntax

```
ares zfilter2 asig, kdamp, kfreq, iM, iN, ib0, ib1, ..., ibM, \
      ia1, ia2, ..., iaN
```

### Initialization

At initialization the number of zeros and poles of the filter are specified along with the corresponding zero and pole coefficients. The coefficients must be obtained by an external filter-design application such as Matlab and specified directly or loaded into a table via *GEN01*. With *zfilter2*, the roots of the characteristic polynomials are solved at initialization so that the pole-control operations can be implemented efficiently.

### Performance

The *filter2* opcodes perform filtering using a transposed form-II digital filter lattice with no time-varying control. *zfilter2* uses the additional operations of radial pole-shearing and angular pole-warping in the Z plane.

Pole shearing increases the magnitude of poles along radial lines in the Z-plane. This has the affect of altering filter ring times. The k-rate variable *kdamp* is the damping parameter. Positive values (0.01 to 0.99) increase the ring-time of the filter (hi-Q), negative values (-0.01 to -0.99) decrease the ring-time of the filter, (lo-Q).

Pole warping changes the frequency of poles by moving them along angular paths in the Z plane. This operation leaves the shape of the magnitude response unchanged but alters the frequencies by a constant factor (preserving 0 and p). The k-rate variable *kfreq* determines the frequency warp factor. Positive values (0.01 to 0.99) increase frequencies toward p and negative values (-0.01 to -0.99) decrease frequencies toward 0.

Since *filter2* implements generalized recursive filters, it can be used to specify a large range of general DSP algorithms. For example, a digital waveguide can be implemented for musical instrument modeling using a pair of *delayr* and *delayw* opcodes in conjunction with the *filter2* opcode.

## Examples

A controllable second-order IIR filter operating on an a-rate signal:

```
a1 zfilter2 asig, kdamp, kfreq, 1, 2, 1, ia1, ia2 ; controllable a-rate ; IIR filter
```

## See Also

*filter2*

## Credits

Author: Michael A. Casey  
M.I.T.  
Cambridge, Mass.  
1997

New in Version 3.47



# zir

zir — Lecture à partir d'une position dans un espace zk au taux-i.

## Description

Lecture à partir d'une position dans un espace zk au taux-i.

## Syntaxe

```
ir zir indx
```

## Initialisation

*indx* -- pointe vers la position zk à lire.

## Exécution

*zir* lit le signal à la position *indx* dans l'espace zk.

## Exemples

Voici un exemple de l'opcode zir. Il utilise le fichier *zir.csd* [examples/zir.csd].

### Exemple 621. Exemple de l'opcode zir.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zir.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple instrument.
instr 1
; Set the zk variable #1 to 32.594.
ziw 32.594, 1
endin

; Instrument #2 -- prints out zk variable #1.
instr 2
; Read the zk variable #1 at i-rate.
```

```
    il zir 1
    ; Print out the value of zk variable #1.
    print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zar, zarg, zkr*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# ziw

ziw — Ecrit dans une variable zk au taux-i sans mixage.

## Description

Ecrit dans une variable zk au taux-i sans mixage.

## Syntaxe

```
ziw isig, indx
```

## Initialisation

*isig* -- Initialise la valeur de la position zk.

*indx* -- Pointe sur la position zk ou za vers laquelle écrire.

## Exécution

ziw écrit *isig* dans la variable zk spécifié par *indx*.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces zk ou za. Sinon, une erreur est rapportée, la valeur 0 est retournée, et il n'y a aucune écriture.

## Exemples

Voici un exemple de l'opcode ziw. Il utilise le fichier *ziw.csd* [examples/ziw.csd].

### Exemple 622. Exemple de l'opcode ziw.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o ziw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple instrument.
instr 1
```

```
; Set zk variable #1 to 64.182.
ziw 64.182, 1
endin

; Instrument #2 -- prints out zk variable #1.
instr 2
; Read zk variable #1 at i-rate.
il zir 1

; Print out the value of zk variable #1.
print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zaw, zawm, ziwm, zkw, zkwm*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# ziwm

ziwm — Ecrit dans une variable zk au taux-i avec mixage.

## Description

Ecrit dans une variable zk au taux-i avec mixage.

## Syntaxe

```
ziwm isig, indx [, imix]
```

## Initialisation

*isig* -- initialise la valeur à la position zk.

*indx* -- pointe sur la position zk vers laquelle écrire.

*imix* (facultatif, par défaut=1) -- indique si le mixage doit avoir lieu.

## Exécution

*ziwm* est un opcode de mixage, il ajoute le signal à la valeur actuelle de la variable. Si aucun *imix* n'est spécifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw* et *zaw*. Toute autre valeur entraînera un mixage.

*Attention* : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm* et *zawm*, il faut faire attention à ce que les variables qui reçoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche serait d'établir certains intervalles de variables zk et za à utiliser pour le mixage, puis d'utiliser *zkcl* ou *zACL* pour effacer ces intervalles.

## Exemples

Voici un exemple de l'opcode *ziwm*. Il utilise le fichier *ziwm.csd* [examples/ziwm.csd].

### Exemple 623. Exemple de l'opcode *ziwm*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ziwm.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple instrument.
instr 1
; Add 20.5 to zk variable #1.
ziwm 20.5, 1
endin

; Instrument #2 -- another simple instrument.
instr 2
; Add 15.25 to zk variable #1.
ziwm 15.25, 1
endin

; Instrument #3 -- prints out zk variable #1.
instr 3
; Read zk variable #1 at i-rate.
il zir 1

; Print out the value of zk variable #1.
; It should be 35.75 (20.5 + 15.25)
print il
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
; Play Instrument #3 for one second.
i 3 0 1
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir Aussi

*zaw, zawm, ziw, zkw, zkwm*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkcl

zkcl — Efface une ou plusieurs variable dans l'espace zk.

## Description

Efface une ou plusieurs variable dans l'espace zk.

## Syntaxe

```
zkcl kfirst, klast
```

## Exécution

*ksig* -- Le signal d'entrée

*kfirst* -- Première position zk ou za de l'intervalle à effacer.

*klast* -- Dernière position zk ou za de l'intervalle à effacer.

*zkcl* efface une ou plusieurs variables dans l'espace zk. Ceci est utile pour les variables utilisées comme accumulateur pour mélanger des signaux de taux-k à chaque cycle, mais qui doivent être effacés avant le prochain groupe de calculs.

## Exemples

Voici un exemple de l'opcode zkcl. Il utilise le fichier *zkcl.csd* [examples/zkcl.csd].

### Exemple 624. Exemple de l'opcode zkcl.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkcl.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Linearly vary a k-rate signal from 220 to 1760.
kline line 220, p3, 1760
```

```

; Add the linear signal to zk variable #1.
zkw kline, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Read zk variable #1.
kfreq zkr 1

; Use the value of zk variable #1 to vary
; the frequency of a sine waveform.
a1 oscil 20000, kfreq, 1

; Generate the audio output.
out a1

; Clear the zk variables, get them ready for
; another pass.
zkcl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
; Play Instrument #2 for three seconds.
i 2 0 3
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zacr, zkwm, zkw, zkmod, zkr*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.



# zkmod

zkmod — Facilite la modulation d'un signal par un autre.

## Description

Facilite la modulation d'un signal par un autre.

## Syntaxe

```
kres zkmod ksig, kzkmod
```

## Exécution

*ksig* -- Le signal d'entrée

*kzkmod* -- contrôle quelle variable zk est utilisée pour la modulation. Une valeur positive signifie une modulation additive, une valeur négative une modulation multiplicative. La valeur 0 ne fait aucun changement à *ksig*. *kzkmod* peut être de taux-i ou de taux-k.

*zkmod* Facilite la modulation d'un signal par un autre, le signal de modulation provenant d'une variable zk. La modulation spécifiée peut être additive ou multiplicative.

## Exemples

Voici un exemple de l'opcode zkmod. Il utilise le fichier *zkmod.csd* [examples/zkmod.csd].

### Exemple 625. Exemple de l'opcode zkmod.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d          ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o zkmod.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Initialize the ZAK space.
; Create 2 a-rate variables and 2 k-rate variables.
zakinit 2, 2

; Instrument #1 -- a signal with jitter.
instr 1
; Generate a k-rate signal goes from 30 to 2,000.
kline line 30, p3, 2000

; Add the signal into zk variable #1.
```

```

    zkw kline, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Create a k-rate signal modulated the jitter opcode.
kamp init 20
kcpmin init 40
kcpmax init 60
kjtr jitter kamp, kcpmin, kcpmax

; Get the frequency values from zk variable #1.
kfreq zkr 1
; Add the the frequency values in zk variable #1 to
; the jitter signal.
kjfreq zkmod kjtr, 1

; Use a simple sine waveform for the left speaker.
aleft oscil 20000, kfreq, 1
; Use a sine waveform with jitter for the right speaker.
aright oscil 20000, kjfreq, 1

; Generate the audio output.
outs aleft, aright

; Clear the zk variables, prepare them for
; another pass.
zkcl 0, 2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zamod, zkcl, zkr, zkwm, zkw*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkr

zkr — Lecture à partir d'une position dans l'espace zk au taux-k.

## Description

Lecture à partir d'une position dans l'espace zk au taux-k.

## Syntaxe

kres **zkr** kndx

## Initialisation

*kndx* -- pointe sur la position za à lire.

## Exécution

*zkr* lit la suite de nombres décimaux à *kndx* dans l'espace zk.

## Exemples

Voici un exemple de l'opcode zkr. Il utilise le fichier *zkr.csd* [examples/zkr.csd].

### Exemple 626. Exemple de l'opcode zkr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Linearly vary a k-rate signal from 440 to 880.
kline line 440, p3, 880

; Add the linear signal to zk variable #1.
zkw kline, 1
endin
```

```
; Instrument #2 -- generates audio output.
instr 2
; Read zk variable #1.
kfreq zkr 1

; Use the value of zk variable #1 to vary
; the frequency of a sine waveform.
a1 oscil 20000, kfreq, 1

; Generate the audio output.
out a1

; Clear the zk variables, get them ready for
; another pass.
zkcl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zar, zarg, zir, zkcl, zkmod, zkwm, zkw*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkw

zkw — Ecrit dans une variable zk au taux-k sans mixage.

## Description

Ecrit dans une variable zk au taux-k sans mixage.

## Syntaxe

**zkw** ksig, kndx

## Exécution

*ksig* -- valeur à écrire dans la position zk.

*kndx* -- pointe sur la position zk ou za vers laquelle écrire.

*zkw* écrit *ksig* dans la variable zk spécifiée par *kndx*.

## Exemples

Voici un exemple de l'opcode *zkw*. Il utilise le fichier *zkw.csd* [exemples/zkw.csd].

### Exemple 627. Exemple de l'opcode *zkw*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Linearly vary a k-rate signal from 100 to 1,000.
kline line 100, p3, 1000

; Add the linear signal to zk variable #1.
zkw kline, 1
endin

; Instrument #2 -- generates audio output.
```

```
instr 2
; Read zk variable #1.
kfreq zkr 1

; Use the value of zk variable #1 to vary
; the frequency of a sine waveform.
a1 oscil 20000, kfreq, 1

; Generate the audio output.
out a1

; Clear the zk variables, get them ready for
; another pass.
zkcl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zaw, zawm, ziw, ziwm, zkr, zkwm*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkwm

zkwm — Ecrit dans une variable zk au taux-k avec mixage.

## Description

Ecrit dans une variable zk au taux-k avec mixage.

## Syntaxe

```
zkwm ksig, kndx [, imix]
```

## Initialisation

*imix* (facultatif) -- indique si le mixage sera fait.

## Ex#cutio#n

*ksig* -- valeur à écrire dans l'espace zk.

*kndx* -- pointe sur la position zk ou za vers laquelle écrire.

*zkwm* est un opcode de mixage, il ajoute le signal à la valeur courante de la variable. Si aucun *imix* n'est spécifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw*, et *zaw*. Toutes autres valeurs entraînera un mixage.

*Avertissement* : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm*, et *zawm*, il faut faire attention à ce que les variables qui reçoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche possible serait d'établir certains intervalles de variables zk ou za à utiliser pour le mixage, puis d'utiliser ensuite *zkcl* ou *zACL* pour effacer ces variables.

## Exemples

Voici un exemple de l'opcode *zkwm*. Il utilise le fichier *zkwm.csd* [examples/zkwm.csd].

### Exemple 628. Exemple de l'opcode zkwm.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkwm.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a basic instrument.
instr 1
; Generate a k-rate signal.
; The signal goes from 30 to 20,000 then back to 30.
kramp linseg 30, p3/2, 20000, p3/2, 30

; Mix the signal into the zk variable #1.
zkwm kramp, 1
endin

; Instrument #2 -- another basic instrument.
instr 2
; Generate another k-rate signal.
; This is a low frequency oscillator.
klfo lfo 3500, 2

; Mix this signal into the zk variable #1.
zkwm klfo, 1
endin

; Instrument #3 -- generates audio output.
instr 3
; Read zk variable #1, containing a mix of both signals.
kamp zkr 1

; Create a sine waveform. Its amplitude will vary
; according to the values in zk variable #1.
al oscil kamp, 880, 1

; Generate the audio output.
out al

; Clear the zk variable, get it ready for
; another pass.
zkcl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 5 seconds.
i 1 0 5
; Play Instrument #2 for 5 seconds.
i 2 0 5
; Play Instrument #3 for 5 seconds.
i 3 0 5
e

</CsScore>
</CsoundSynthesizer>

```

## Voir Aussi

*zaw, zawm, ziw, ziwm, zkcl, zkw, zkr*

## Crédits

Auteur : Robin Whittle  
Australie



Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

---

# Instructions de Partition et Routines GEN

## Instructions de Partition

Les instructions utilisées dans les partitions sont :

- *a* - Avance le temps de la partition d'une quantité spécifiée
- *b* - Réinitialise l'horloge
- *e* - Marque la fin de la dernière section de la partition
- *f* - Appelle une *routine GEN* pour placer des valeurs dans une table de fonction stockée
- *i* - Active un instrument à une date spécifique et pour une certaine durée
- *m* - Positionne une marque nommée dans la partition
- *n* - Répète une section marquée
- *q* - Rend un instrument silencieux
- *r* - Commence une section répétée
- *s* - Marque la fin d'une section
- *t* - Fixe le tempo
- *v* - Permet une modification temporelle variable localement des événements de la partition
- *x* - Ignore le reste de la section courante
- *{* - Commence une boucle imbriquable, sans section
- *}* - Termine une boucle imbriquable, sans section

# Instruction a (ou Instruction Avancer)

a — Avancer le temps de la partition de la quantité spécifiée.

## Description

Provoque l'avancement du temps de la partition de la quantité spécifiée sans produire d'échantillons sonores.

## Syntaxe

a p1 p2 p3

## Exécution

p1	Non significatif. Habituellement zéro.
p2	Date en pulsations à laquelle l'avance doit commencer.
p3	Nombre de pulsations duquel il faut avancer sans produire de son.
p4	
p5	Non significatifs.
p6	
.	
.	

## Considérations Spéciales

Cette instruction permet d'avancer le compteur de pulsations dans une partition sans générer les échantillons sonores correspondants. On peut l'utiliser quand une section de la partition est incomplète (le début ou le milieu sont manquants) et que l'on ne souhaite pas générer et écouter une longue période de silence.

p2, date d'activation, et p3, nombre de pulsations, sont traités comme dans l'*instruction i*, en tenant compte du tri et des modifications par les *instructions t*.

Une *instruction a* sera insérée temporairement dans la partition par la fonction Score Extract lorsque l'extrait commence après le début de la Section. Ceci afin de conserver le compte de pulsations de la partition originale pour les messages de pic d'amplitude qui sont rapportés sur la console de l'utilisateur.

A chaque exécution d'un orchestre lorsqu'une *instruction a* est rencontrée, sa présence et son effet son rapportés sur la console de l'utilisateur.

# Instruction b

b — Cette instruction réinitialise l'horloge.

## Description

Cette instruction réinitialise l'horloge.

## Syntaxe

**b** p1

## Exécution

p1 -- Spécifie comment l'horloge doit être réglée.

## Considérations Spéciales

p1 est le nombre de pulsations par lequel les valeurs p2 des *instructions i* suivantes sont modifiées. Si p1 est positif, l'horloge est avancée, et les notes suivantes apparaissent plus tard, le nombre de pulsations spécifié par p1 étant ajouté au p2 des notes. Si p1 est négatif, l'horloge est retardée, et les notes suivantes apparaissent plus tôt, le nombre de pulsations spécifié par p1 étant soustrait du p2 des notes. L'effet n'est pas cumulatif. L'horloge est réinitialisée avec chaque *instruction b*. Si p1 = 0, l'horloge revient à sa position initiale, et les notes suivantes apparaissent à leur position spécifiée en p2.

## Exemples

```
i1      0      2
i1      10     888

b 5
i2      1      1      440      ; "avance" l'horloge
i2      2      1      480      ; date de début = 6
                                   ; date de début = 7

b -1
i3      3      2      3.1415    ; retarde l'horloge
i3      5.5    1      1.1111    ; date de début = 2
                                   ; date de début = 4.5

b 0
i4      10     200    7          ; réinitialise l'horloge à la normale
                                   ; date de début = 10
```

## Crédits

Explication suggérée et exemple fourni par Paul Winkler. (Version 4.07 de Csound)

# Instruction e

e — On peut utiliser cette instruction pour marquer la fin de la dernière section de la partition.

## Description

On peut utiliser cette instruction pour marquer la fin de la dernière section de la partition.

## Syntaxe

e [ temps ]

## Exécution

Le premier p-champ *temps* est facultatif et s'il est présent, il détermine la date de fin (en pulsations) de la dernière section de la partition. Cette date doit être après le dernier événement sinon elle n'aura pas d'effet. Les instruments "actifs en permanence" se termineront à cette date. Cette manière d'allonger la section est utile pour éviter les coupures prématurées de chute de réverbération ou d'autres effets.

## Considérations Spéciales

L'*instruction e* est contextuellement identique à une *instruction s*. De plus, l'*instruction e* termine toute génération de signal (y compris une exécution indéfinie) et ferme tous les fichiers d'entrée et de sortie.

Si une *instruction e* intervient avant la fin de la partition, toutes les lignes suivantes de la partition seront ignorées.

Dans un fichier de partition pas encore trié, l'*instruction e* est facultative. Si un fichier de partition n'a pas d'*instruction e*, alors la fonction Sort en fournira une.

# Instruction f (ou Instruction de Table de Fonction)

f — Provoque l'écriture de valeurs dans une table de fonction en mémoire par une routine GEN.

## Description

Provoque l'écriture de valeurs dans une table de fonction en mémoire par une routine GEN pour utilisation par des instruments.

## Syntaxe

**f** p1 p2 p3 p4 p5 ... PMAX

## Exécution

*p1* -- Numéro de table sous lequel la fonction mémorisée sera connue. Un nombre négatif signifie une demande de destruction de la table.

*p2* -- Date d'activation de la génération de la fonction (ou de sa destruction) en pulsations.

*p3* -- Taille de la table de la fonction (c'est-à-dire nombre de points). Doit être une puissance de 2, ou une puissance de 2 plus 1 si ce nombre est positif. La taille de table maximale est de 16777216 ( $2^{24}$ ) points.

*p4* -- Numéro de la routine GEN à appeler (voir *ROUTINES GEN*). Une valeur négative supprimera la normalisation.

*p5* ... *PMAX* -- Paramètres dont la signification est déterminée par la routine GEN particulière.

## Considérations Spéciales

Les tables de fonction sont des tableaux de valeurs en virgule flottante. On peut créer une simple onde sinusoïdale avec cette ligne :

```
f 1 0 1024 10 1
```

Cette table utilise *GEN10* pour son remplissage.

Historiquement, à cause des contraintes des anciennes plates-formes, Csound ne pouvait accepter que des tables dont la taille était une puissance de deux. Cette limitation a été levée dans les récentes versions, et l'on peut créer des tables de n'importe quelle taille. Cependant, pour créer une table dont la taille n'est pas une puissance de deux (ou une puissance de deux plus un), il faut spécifier la taille comme un nombre négatif.



### Note

Il y a des opcodes qui n'accepteront pas des tables dont la taille n'est pas une puissance de deux, car ils comptent sur cela pour leur optimisation interne.

Pour les tableaux dont la longueur est une puissance de 2, l'allocation d'espace mémoire est toujours pré-

vue pour  $2^n$  points plus un *point de garde*. La valeur du point de garde, utilisée pour la lecture avec interpolation, peut être fixée automatiquement selon le but de la table : si la *taille* est une puissance de 2 exacte, le point de garde sera une copie du premier point ; cela convient pour la *lecture cyclique avec interpolation* comme dans *oscili*, etc., et devrait même être utilisé pour la version sans interpolation *oscil* pour rester consistant. Si la *taille* est fixée à  $2^n + 1$ , le point de garde prolongera automatiquement le contour des valeurs de la table ; cela convient pour les fonctions à lecture non-cyclique comme dans *envplx*, *oscil1*, *oscil1i*, etc.

La taille de la table est utilisée comme un code pour indiquer à Csound comment remplir ce point de garde. Si la taille est exactement une puissance de deux, alors le point de garde contient une copie du premier point de la table. Si la taille est une puissance-de-deux plus un, Csound étend le contour de la fonction stockée dans la table pour un point supplémentaire.

Les tables sont allouées dans la mémoire primaire, avec les données d'instrument. Le nombre maximum de tables était limité à 200. Ceci a changé et il n'est plus limité que par la quantité de mémoire disponible. (Actuellement il y a une limitation logicielle de 300, qui est augmentée automatiquement selon les besoins).

On peut supprimer une table de fonction existante par une *instruction f* contenant un p1 négatif et une date d'activation adéquate. Une table de fonction est également supprimée par la génération d'une autre table avec le même p1. Les fonctions ne sont pas automatiquement effacées à la fin d'une section de partition.

La date p2 est traitée de la même manière que dans l'*instruction i* en tenant compte du tri et des modifications par les *instructions t*. Si une *instruction f* et une *instruction i* ont le même p2, le tri donnera la priorité à l'*instruction f* afin que le table de fonction soit disponible pendant l'initialisation de la note.



## Avertissement

Le nombre maximum de p-champs acceptés dans la partition est déterminé par PMAX (une variable de compilation). PMAX vaut actuellement 1000. Cela peut éliminer des valeurs entrées au moyen de *GEN02*. Pour contourner cette limitation, utiliser *GEN23* ou *GEN28* pour lire les valeurs à partir d'un fichier.

On peut utiliser une *instruction f 0* (avec zéro en p1 et p2 positif) pour créer une date sans action associée. De tels marqueurs temporels sont utiles pour remplir une section de partition (voir l'*instruction s*) et pour lancer une exécution de Csound à partir d'événements en temps réel (par exemple en n'utilisant que des entrées MIDI sans événements de partition). La durée indique le nombre de secondes de l'exécution de Csound. Si l'on veut que Csound tourne pendant 10 heures, on utilisera :

```
f0 36000
```

La manière la plus simple de remplir une table (f1) avec des 0 est :

```
f1 0 xx 2 0
```

where xx = table size.

La manière la plus simple de remplir une table (f1) avec n'importe quelle valeur unique est :

```
f1 0 xx -7 yy xx yy
```

où xx = taille de la table et yy = n'importe quelle valeur unique

Dans les deux exemple ci-dessus, la taille de la table (p3) doit être une puissance de 2 ou une puissance-de-2 + 1.

## Voir aussi

*ROUTINES GEN*

## Crédits

Mise à jour en août 2002 grâce à une note de Rasmus Ekman. Il n'y a plus de limite codée en dur à 200 tables de fonction.



# Instruction i (Instruction d'Instrument ou de Note)

i — Active un instrument à une date précise et pour une certaine durée.

## Description

Cette instruction est nécessaire pour activer un instrument à une date précise et pour une certaine durée. Les valeurs des champs de paramètre sont passées à cet instrument avant son initialisation, et demeurent valides durant toute son exécution.

## Syntaxe

i p1 p2 p3 p4 ...

## Initialisation

*p1* -- Numéro d'instrument, habituellement un nombre entier non négatif. Une partie décimale facultative permet d'ajouter une étiquette indiquant des liaisons entre des notes particulières d'aggrégats consécutifs. Un *p1* négatif (incluant une étiquette) peut être utilisé pour faire cesser une note « tenue » particulière.

*p2* -- Date de début en unités arbitraires appelées pulsations.

*p3* -- Durée en pulsations (habituellement positive). Une valeur négative démarre une note tenue (voir aussi *ihold*). On peut aussi utiliser une valeur négative pour les instruments 'toujours actifs' comme la réverbération. Ces notes ne sont pas terminées par des *instruction s*. Une valeur nulle provoquera une passe d'initialisation sans exécution (voir aussi *instr*).

*p4* ... -- Paramètres dont la signification est déterminée par l'instrument.

## Exécution

Une pulsation vaut une seconde, à moins qu'il n'y ait une *instruction t* dans cette section de la partition ou une *option -t* dans la ligne de commande.

Les dates de début ou d'action sont relatives au début d'une section (voir l'*instruction s*), qui reçoit la date 0.

Dans une section, les instructions de note peuvent être placées dans n'importe quel ordre. Avant d'être envoyées à l'orchestre, les instructions non triées de la partition doivent être traitées par la fonction Sort, qui les ordonnera par valeurs de *p2* croissantes. Les notes ayant la même valeur en *p2* seront triées par *p1* croissants ; si elles ont le même *p1*, alors par *p3* croissants.

Les notes peuvent être superposées, c'est-à-dire qu'un seul instrument peut jouer n'importe quel nombre de notes simultanément. (Les copies nécessaires de l'espace de données de l'instrument seront allouées dynamiquement par le chargeur de l'orchestre). Chaque note se termine normalement à la fin de sa durée en *p3*, ou à la réception d'un signal MIDI noteoff. Un instrument peut modifier sa propre durée en changeant la valeur de son *p3* pendant l'initialisation de la note, ou en se prolongeant lui-même par l'action d'une unité *linenr* ou *xtratim*.

Un instrument peut être activé et réglé pour une durée indéfinie soit en lui donnant un *p3* négatif soit en incluant un *ihold* dans le code de son temps-*i*. Si une note tenue est active, une *instruction i* avec un *p1*

*correspondant* ne provoquera pas une nouvelle allocation mais prendra l'espace de données de la note tenue. Les nouveaux p-champs (y compris p3) seront maintenant effectifs, et une passe de temps-i sera exécutée pendant laquelle les unités peuvent être soit initialisées à nouveau soit autorisées à continuer comme requis pour une note liée (voir *tigoto*). Une note tenue peut être suivie soit par une autre note tenue soit par une note de durée finie. Une note tenue continuera à être jouée au-delà des fins de section (voir l'*instruction s*). Elle est arrêtée seulement par un *turnoff* ou par une *instruction i* avec un p1 négatif correspondant ou par une *instruction e*.

Il est possible d'avoir plusieurs instances (habituellement, mais pas forcément, des notes de hauteurs différentes) du même instrument, tenues simultanément, via des valeurs négatives de p3. L'instrument peut ensuite recevoir de nouveaux paramètres de la partition. C'est utile pour éviter de longs *linseg* codés en dur, et peut être accompli en ajoutant une partie décimale au numéro de l'instrument.

Par exemple, pour tenir trois copies de l'instrument 10 dans un accord :

```
i10.1  0  -1  7.00
i10.2  0  -1  7.04
i10.3  0  -1  7.07
```

Les instructions *i* suivantes peuvent faire référence aux mêmes instances de note active, et si la définition de l'instrument est faite proprement, les nouveaux p-champs peuvent servir à changer le caractère des notes jouées. Par exemple, pour faire glisser l'accord précédent d'une octave vers le haut et le laisser résonner :

```
i10.1  1  1  8.00
i10.2  1  1  8.04
i10.3  1  1  8.07
```



## Astuce

Pour la terminaison des notes, il faut tenir compte du fait que  $i\ 1.1 == i\ 1.10$  et que  $i\ 1.1 != i\ 1.01$ . Le nombre maximum de positions décimales que l'on peut utiliser dépend de la précision avec laquelle Csound a été compilé (Voir *Csound Double (64 bit)* ou *Float (32 bit)*)

La définition de l'instrument doit prendre ceci en compte, cependant, spécialement si l'on veut éviter les clics (voir l'exemple ci-dessous).

Noter que la notation décimale du numéro d'instrument ne peut pas être utilisée en conjonction avec le MIDI en temps réel. Dans ce cas, l'instrument serait monodique tant qu'une note est tenue.

Les notes liées à des instances précédentes du même instrument, devraient éviter la plus grande partie de l'initialisation au moyen de *tigoto*, sauf pour les valeurs entrées dans la partition. Par exemple, tous les opcodes de lecture de table dans l'instrument, seront habituellement sautés en initialisation, car ils mémorisent en interne leur phase. Si celle-ci est brutalement modifiée, on entendra des clics en sortie.

Noter que plusieurs opcodes (comme *delay* et *reverb*) sont prévus pour une initialisation facultative. Pour utiliser cette possibilité, l'*opcode tival* est approprié. Ainsi, il n'y a pas besoin de les escamoter par un saut *tigoto*.

A partir de la version 3.53 de Csound, les chaînes sont reconnues dans les p-champs des opcodes qui les acceptent (*convolve*, *adsyn*, *diskin*, etc.). Il ne peut y avoir qu'une seule chaîne par ligne de la partition.

On peut aussi terminer une note depuis la partition en utilisant un nombre négatif pour l'instrument (p1). Cela équivaut à utiliser l'opcode *turnoff2*. Lorsqu'une note est terminée depuis la partition, elle peut

avoir un relâchement (si *xtratim* ou des opcodes avec une section de relâchement tels que *linenr* sont utilisés) et seules les notes ayant la même partie fractionnaire sont arrêtées. De plus, seule la dernière instance de l'instrument est arrêtée, si bien qu'il faut autant de numéros d'instrument négatifs que de numéros positifs pour que toutes les notes soient arrêtées.

```
i 1.1 1 300 8.00
i 1.2 1 300 8.04
i 1.3 1 -300 8.07
i 1.3 1 -300 8.09

; noter que les p-champs suivant p2 sont ignorés
; si le numéro d'instrument est négatif
i -1.1 3 1 4.00
i -1.2 4 51 4.04
i -1.3 5 1 4.07
i -1.3 6 10 4.09
```

## Considérations Spéciales

Le numéro d'instrument maximum était de 200. Cela a changé et il n'est plus limité que par la capacité mémoire (actuellement, il y a une limite logicielle de 200 ; celle-ci est étendue automatiquement si nécessaire).

## Exemples

Voici un instrument capable de découvrir s'il est lié à une note précédente (*tival* retourne 1), et s'il doit être tenu (p3 négatif). L'attaque et la chute sont traitées en conséquence :

```
instr 10

icps    init    cpspch(p4)          ; Reçoit la hauteur cible de l'évènement de partition
iporime init    abs(p3)/7           ; La durée du portamento dépend de celle de la note
iamp0   init    p5                  ; Fixe l'amplitude par défaut
iamp1   init    p5
iamp2   init    p5

itie    tival
if itie == 1      igoto nofadein    ; Teste si cette note est liée,
iamp0   init      0                 ; si non alors entrée progressive

nofadein:
if p3 < 0         igoto nofadeout   ; Teste si cette note est tenue,
iamp2   init      0                 ; si non alors disparition progressive

nofadeout:
; Maintenant générer l'amplitude à partir des valeurs fixées :
kamp    linseg    iamp0, .03, iamp1, abs(p3)-.03, iamp2

; Passe le reste de l'initialisation pour une note liée :
tigoto   tieskip

kcps     init     icps              ; Initialise la hauteur pour une note non liée
kcps     port     icps, iporime, icps ; Glisse vers la hauteur cible

kpwr     oscil    .4, rnd(1), 1, rnd(.7) ; Un oscillateur simple en dent de scie
ar       vco      kamp, kcps, 3, kpwr+.5, 1, 1/icps

; (Utilisé pour tester - on peut fixer ipch à cpspch(p4+2)
; et voir le spectre en sortie)
; ar oscil kamp, kcps, 1

out      ar

tieskip:                                     ; Passe certaines initialisations pour une note liée
endin
```

Une simple partition avec trois instances de l'instrument ci-dessus :

```
f1  0 8192 10 1          ; Sinus
i10.1  0   -1   7.00    10000
i10.2  0   -1   7.04
i10.3  0   -1   7.07
i10.1  1   -1   8.00
i10.2  1   -1   8.04
i10.3  1   -1   8.07
i10.1  2    1   7.11
i10.2  2    1   8.04
i10.3  2    1   8.07
e
```

## Crédits

Texte supplémentaire (Version 4.07 de Csound) expliquant les notes liées, publié par Rasmus Ekman d'après une note de David Kirsh, postée sur la liste de courrier électronique de Csound. Instrument en exemple par Rasmus Ekman.

Mise à jour Août 2002 grâce à une note de Rasmus Ekman. Il n'y a plus de limite codée en dur à 200 instruments.

# Instruction m (Instruction de Marquage)

m — Positionne une marque nommée dans la partition.

## Description

Positionne une marque nommée dans la partition, qui peut être utilisée par une *instruction n*.

## Syntaxe

m pl

## Initialisation

pl -- Nom de la marque.

## Exécution

Peut être utile pour construire une structure couplet refrain dans la partition. Les noms peuvent contenir des lettres et des chiffres.

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Nouveau dans la version 3.48 de Csound

# Instruction n

n — Répète une section.

## Description

Répète une section depuis l'*instruction m* référencée.

## Syntaxe

n p1

## Initialisation

p1 -- Nom de la marque à répéter.

## Exécution

Peut-être utile pour construire une structure couplet refrain dans la partition. Les noms peuvent contenir des lettres et des chiffres.

Par exemple, la partition suivante :

```
m foo
il 0 1
il 1 1.5
il 2.5 2
s
il 0 10
s
n foo
e
```

Sera transmise par le préprocesseur à Csound comme :

```
il 0 1
il 1 1.5
il 2.5 2
s
il 0 10
s
;; ceci est la section nommée répétée
il 0 1
il 1 1.5
il 2.5 2
s
;; fin de la section nommée
e
```

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Nouveau dans la version 3.48 de Csound

# Instruction q

q — Cette instruction peut être utilisée pour rendre un instrument silencieux.

## Description

Cette instruction peut être utilisée pour rendre un instrument silencieux.

## Syntaxe

**q** *p1* *p2* *p3*

## Exécution

*p1* -- Numéro de l'instrument à rendre muet/sonore.

*p2* -- Date d'action en pulsations.

*p3* -- Détermine si l'instrument doit être rendu silencieux ou sonore. La valeur 0 signifie silencieux, toute autre valeur signifie sonore.

Noter que ceci n'affecte pas les instruments déjà actifs à la date *p2*. Ça bloque toute tentative d'en démarrer un après cette date.



# Instruction r (Instruction Répéter)

r — Débute une section répétée.

## Description

Débute une section répétée, qui dure jusqu'à la prochaine instruction *s*, *r* ou *e*.

## Syntaxe

**r** p1 p2

## Initialisation

*p1* -- Nombre de répétitions de la section demandé.

*p2* -- Macro(nom) pour indexer chaque répétition (facultatif).

## Exécution

Afin de rendre les sections plus souples qu'une simple édition, la macro nommée en *p2* reçoit la valeur 1 à la première boucle dans la section, 2 à la seconde, 3 à la troisième, etc. On peut l'utiliser pour changer la valeur des p-champs, ou l'ignorer.



### Avertissement

A cause de sérieux problèmes d'interaction avec l'expansion de macro, les sections doivent commencer et finir dans le même fichier, à l'extérieure de toute macro.

## Exemples

Voici un exemple d'instruction r. Il utilise le fichier *r.sco* [examples/r.csd].

### Exemple 629. Exemple d'instruction r.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o r.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; The score's p4 parameter has the number of repeats.
```

```
kreps = p4
; The score's p5 parameter has our note's frequency.
kcps = p5

; Print the number of repeats.
printks "Repeated %i time(s).\n", 1, kreps

; Generate a nice beep.
a1 oscil 20000, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; We'll repeat this section 6 times. Each time it
; is repeated, its macro REPS_MACRO is incremented.
r6 REPS_MACRO

; Play Instrument #1.
; p4 = the r statement's macro, REPS_MACRO.
; p5 = the frequency in cycles per second.
i 1 00.10 00.10 $REPS_MACRO 1760
i 1 00.30 00.10 $REPS_MACRO 880
i 1 00.50 00.10 $REPS_MACRO 440
i 1 00.70 00.10 $REPS_MACRO 220

; Marks the end of the section.
s

e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Nouveau dans la version 3.48 de Csound

Exemple écrit par Kevin Conder

# Instruction s

s — Marque le fin d'une section.

## Description

L'*instruction s* marque le fin d'une section.

## Syntaxe

**s** [ temps ]

## Initialisation

Le premier p-champ *temps* est facultatif et s'il est présent, il détermine la date de fin (en pulsations) de la section. Cette date doit être après la fin du dernier évènement de la section sinon elle n'aura pas d'effet. On peut l'utiliser pour créer une pause avant le début de la section suivante ou pour permettre aux instruments "actifs en permanence" tels que les effets de jouer seuls pendant une certaine durée.

## Exécution

Le tri des *instructions i*, des *instructions f* et des *instructions a* par date d'action est effectué section par section.

La modification temporelle par l'*instruction t* est faite section par section.

Toutes les dates d'action à l'intérieur d'une section sont relatives à son début. Une instruction de section établit un nouveau temps relatif de 0, mais n'a pas d'autres effets de réinitialisation (par exemple les tables de fonction mémorisées sont préservées par delà les limites de section).

On considère qu'une section est complète lorsque toutes les dates d'action et toutes les durées finies ont été satisfaites. (C'est-à-dire que la "longueur" d'une section est déterminée par la dernière action apparue ou par l'arrêt du système). Une section peut être allongée par l'utilisation d'une *instruction f0* ou en fournissant la valeur de *p* facultative à l'*instruction s*.

A la fin d'une section, le système provoque automatiquement le nettoyage des instruments inactifs et de leur espace de données.



### Note

- Puisque les instructions de partition sont traitées section par section, la quantité de mémoire requise dépend du nombre maximum d'instructions de partition dans une section. L'allocation de mémoire est dynamique, et l'utilisateur sera informé chaque fois que des blocs de mémoire supplémentaires sont demandés pendant le traitement de la partition.
- Pour la dernière section d'une partition, l'*instruction s* est facultative ; l'*instruction e* peut être utilisée à la place.

# Instruction t (Instruction de Tempo)

t — Fixe le tempo.

## Description

Cette instruction fixe le tempo et spécifie les *accelerando* et les *ritardando* de la section courante. Ceci est réalisé en convertissant les pulsations en secondes.

## Syntaxe

t p1 p2 p3 p4 ... (illimité)

## Initialisation

p1 -- Doit être zéro.

p2 -- Tempo initial en pulsations par minute.

p3, p5, p7,... -- Dates en pulsations (en ordre non décroissant).

p4, p6, p8,... -- Tempi pour les dates en pulsations référencées.

## Exécution

Les dates et le Tempo pour chaque date sont donnés en couples ordonnés qui définissent des points sur un graphe « date, tempo ». (L'axe du temps est ici en pulsations et n'est donc pas nécessairement linéaire). Le taux de pulsations d'une section peut être pensé comme un mouvement d'un point à un autre de ce graphe : un mouvement entre deux points à la même hauteur signifie un tempo constant, tandis qu'un mouvement entre deux points de hauteurs différentes traduit un *accelerando* ou un *ritardando* selon le cas. Le graphe peut contenir des discontinuités : deux points ayant la même date mais des tempi différents provoqueront un changement de tempo instantané.

Le mouvement entre différents tempi sur des durées non nulles est inversement linéaire. Cela veut dire qu'un *accelerando* entre deux tempi M1 et M2 procède par interpolation linéaire des durées de chaque pulsation entre 60/M1 et 60/M2.

Le premier tempo doit être donné pour la pulsation 0.

Une fois assigné, un tempo sera effectif à partir de cette date à moins d'être influencé par un tempo suivant, ainsi, le dernier tempo spécifié sera actif jusqu'à la fin de la section.

Une *instruction t* ne s'applique que dans la section dans laquelle elle apparaît. Une seule *instruction t* est pertinente dans une section ; elle peut être placée n'importe où dans la section. Si une section de partition ne contient pas d'*instruction t*, les pulsations sont alors interprétées comme des secondes (c'est-à-dire avec une *instruction t 0 60* implicite).

Nota Bene. Si la commande de Csound comprend une *option -t*, le tempo interprété de toutes les *instruction t* de la partition sera remplacé par le tempo de la ligne de commande.

# Instruction v

v — Permet une modification temporelle variable localement des évènements de la partition.

## Description

L'*instruction v* permet une modification temporelle variable localement des évènements de la partition.

## Syntaxe

**v** p1

## Initialisation

p1 -- facteur de modification temporelle (doit être positif).

## Exécution

L'*instruction v* prend effet avec l'*instruction i* qui la suit, et reste effective jusqu'à la prochaine *instruction v*, *instruction s*, ou *instruction e*.

## Exemples

La valeur de p1 est utilisée comme un coefficient multiplicatif de la date de début (p2) des *instructions i* suivantes.

```
i1  0 1  ; note1
v2
i1  1 1  ; note2
```

Dans cet exemple, la deuxième note apparaît deux pulsations après la première note, et elle est deux fois plus longue.

Bien que l'*instruction v* soit semblable à l'*instruction t*, l'*instruction v* agit localement. Cela veut dire que v n'affecte que les notes suivantes, et que son effet peut être annulé ou changé par une autre *instruction v*.

Les valeurs reportées ne sont pas affectées par l'*instruction v* (voir *Carry*).

```
i1  0 1  ; note1
v2
i1  1 .  ; note2
i1  2 .  ; note3
v1
i1  3 .  ; note4
i1  4 .  ; note5
e
```

Dans cet exemple, note3 et note5 sont jouées simultanément, tandis que note4 est jouée avant note3, c'est-à-dire à sa place initiale. Les durées sont inchangées.

```
i1  0 1
v2
i.  + .
i.  . .
```

Dans cet exemple, l'*instruction v* n'a aucun effet.

# Instruction x

x — Ignore le reste de la section courante.

## Description

On peut utiliser cette instruction pour ignorer le reste de la section courante.

## Syntaxe

**x** valeurbidon

## Initialisation

Tous les p-champs sont ignorés.

# Instruction {

{ — Commence une boucle imbriquable, sans section.

## Description

On peut utiliser les *instructions* { et } pour répéter un groupe d'instructions de partition. Ces boucles ne constituent pas des sections de partition indépendantes et peuvent ainsi répéter des événements dans la même section. Plusieurs boucles peuvent se chevaucher dans le temps ou être imbriquées.

## Syntaxe

{ p1 p2

## Initialisation

p1 -- Nombre de répétitions de la boucle.

p2 -- Un nom de macro qui est automatiquement défini au début de la boucle et dont la valeur est incrémentée à chaque répétition (facultatif). La valeur initiale est zéro et la valeur finale est (p1 - 1).

## Exécution

L'*instruction* { est utilisée conjointement avec l'*instruction* } pour définir des groupes d'événements de partition qui se répètent. Une boucle de partition commence par l'*instruction* { qui définit le nombre de répétitions et un nom de macro unique qui contiendra le compteur de boucle. Le corps d'une boucle peut contenir n'importe quel nombre d'événements (y compris des sauts de section) et il se termine par une *instruction* } ayant sa propre ligne. L'*instruction* } ne prend pas de paramètre.

Le terme "boucle" n'implique aucune sorte de succession temporelle pour les itérations de la boucle. Autrement dit, les valeurs p2 des événements à l'intérieur de la boucle ne sont pas incrémentées automatiquement de la longueur de la boucle à chaque répétition. C'est un avantage car cela permet de définir facilement des groupes d'événements simultanés. La macro de boucle peut être utilisée avec des *expressions de partition* pour incrémenter les dates de début d'événements ou pour faire varier les événements de toute autre manière désirée à chaque répétition. Noter que à la différence de l'*instruction* r, la valeur de la macro au premier passage dans la boucle est zéro (0), pas un (1). Ainsi la valeur finale est inférieure d'une unité au nombre de répétitions.

Les boucles de partition sont un outil très puissant. Bien que semblables à l'outil de répétition de section (l'*instruction* r), leur principal avantage est que les événements de partition dans les itérations successives de la boucle ne sont pas séparés par une fin de section. Ainsi, il est possible de créer plusieurs boucles qui se chevauchent dans le temps. Les boucles peuvent aussi être imbriquées jusqu'à une profondeur de 39 niveaux.



### Avertissement

En raison de sérieux problèmes d'interaction avec l'expansion de macro, les boucles doivent commencer et se terminer dans le même fichier, et pas à l'intérieur d'une macro.

## Exemples



Voici quelques exemples des *instructions* { et }.

### Exemple 630. Répétition séquentielle d'une phrase de trois notes, quatre fois.

```
{ 4 CNT
i1 [0.00 + 0.75 * $CNT.] 0.2 220
i1 [0.25 + 0.75 * $CNT.] . 440
i1 [0.50 + 0.75 * $CNT.] . 880
}
```

interprété comme

```
i1 0.00 0.2 220
i1 0.25 . 440
i1 0.50 . 880

i1 0.75 0.2 220
i1 1.00 . 440
i1 1.25 . 880

i1 1.50 0.2 220
i1 1.75 . 440
i1 2.00 . 880

i1 2.25 0.2 220
i1 2.50 . 440
i1 2.75 . 880
```

### Exemple 631. Création d'un groupe d'harmoniques simultanés.

Dans cet exemple,  $p\mathcal{A}$  contient la fréquence de la note.

```
{ 8 PARTIAL
i1 0 1 [100 * ($PARTIAL. + 1)]
}
```

interprété comme

```
i1 0 1 100
i1 0 1 200
i1 0 1 300
i1 0 1 400
i1 0 1 500
i1 0 1 600
i1 0 1 700
i1 0 1 800
```

Voici un exemple complet des *instructions* { et }. Il utilise le fichier *leftbrace.csd* [examples/left-brace.csd].

### Exemple 632. Un exemple de boucles imbriquées pour créer plusieurs clusters inharmónicos de sinus.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
nchnls = 2

gaReverbSend init 0

; a simple sine wave partial
instr 1
  idur =      p3
  iamp =      p4
  ifreq =     p5
  aenv linseg 0.0, 0.1*idur, iamp, 0.6*idur, iamp, 0.3*idur, 0.0
  aosc oscili aenv, ifreq, 1
  vincr gaReverbSend, aosc
endin

; global reverb instrument
instr 2
  al, ar reverbsc gaReverbSend, gaReverbSend, 0.85, 12000
          outs   gaReverbSend+al, gaReverbSend+ar
          clear   gaReverbSend
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1

{ 4 CNT
  { 8 PARTIAL
    ; start time      duration      amplitude      frequency

    i1 [0.5 * $CNT.] [1 + ($CNT * 0.2)] [500 + (~ * 200)] [800 + (200 * $CNT.) + ($PARTIAL. * 20)]
  }
}

i2 0 6
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 3.52 (?) de Csound. (Fixé dans la version 5.08).

# Instruction }

} — Termine une boucle imbriquable, sans section.

## Description

On peut utiliser les *instructions { et }* pour répéter un groupe d'instructions de partition. Ces boucles ne constituent pas des sections de partition indépendantes et peuvent ainsi répéter des évènements dans la même section. Plusieurs boucles peuvent se chevaucher dans le temps ou être imbriquées.

## Syntaxe

}

## Initialisation

Tous les p-champs sont ignorés.

## Exécution

L'*instruction }* est utilisée conjointement avec l'*instruction {* pour définir des groupes d'évènements de partition qui se répètent. Une boucle de partition commence par l'*instruction {* qui définit le nombre de répétitions et un nom de macro unique qui contiendra le compteur de boucle. Le corps d'une boucle peut contenir n'importe quel nombre d'évènements (y compris des sauts de section) et il se termine par une *instruction }* ayant sa propre ligne. L'*instruction }* ne prend pas de paramètre.

Voir la documentation de l'*instruction {* pour plus de détails.

## Exemples

Voir les exemples de l'article sur l'*instruction {*.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 3.52 (?) de Csound. (Fixé dans la version 5.08).

## Routines GEN

Les routines GEN sont utilisées comme générateurs de données pour les tables de fonction. Quand une table de fonction est créée au moyen de l'*instruction de partition f* la fonction GEN est donnée dans le quatrième argument. Un numéro de GEN négatif implique que la fonction ne sera pas normalisée et qu'elle gardera ses valeurs originales.

## Générateurs Sinus/Cosinus :

- *GEN09* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.

- *GEN10* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.
- *GEN11* - Ensemble additif de partiels cosinus.
- *GEN19* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.
- *GEN30* - Génère des partiels harmoniques en analysant une table existante.
- *GEN33* - Génère des formes d'onde complexes en mélangeant des sinus.
- *GEN34* - Génère des formes d'onde complexes en mélangeant des sinus.

## Générateurs par Morceaux de Ligne/Exponentielle

- *GEN05* - Construit des fonctions à partir de morceaux de courbes exponentielles.
- *GEN06* - Génère une fonction composée de morceaux de polynômes cubiques.
- *GEN07* - Construit des fonctions à partir de morceaux de lignes droites.
- *GEN08* - Génère une courbe spline cubique par morceaux.
- *GEN16* - Crée une table depuis une valeur initiale jusqu'à une valeur terminale.
- *GEN25* - Construit des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).
- *GEN27* - Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

## Routines GEN d'Accès Fichier :

- *GEN01* - Transfère des données d'un fichier son dans une table de fonction.
- *GEN23* - Lit des valeurs numériques à partir d'un fichier texte.
- *GEN28* - Lit un fichier texte qui contient une trajectoire paramétrée par le temps.
- *GEN49* - Transfère les données d'un fichier son MP3 dans une table de fonction.

## Routines GEN d'Accès à des Valeurs Numériques

- *GEN02* - Transfère les données des p-champs dans une table de fonction.
- *GEN17* - Crée une fonction en escalier à partir des paires x-y données.
- *GEN52* - Crée une table multi-canaux entrelacés à partir des tables source indiquées, dans le format attendu par l'opcode *fconv*.

## Routines GEN de Fonction Fenêtre

- *GEN20* - Génère les fonctions de différentes fenêtres.

## Routines GEN de Fonction Aléatoire

- *GEN21* - Génère les tables de différentes distributions aléatoires.
- *GEN40* - Génère une distribution aléatoire à partir d'un histogramme.
- *GEN41* - Génère une liste aléatoire de paires numériques.
- *GEN42* - Génère une distribution aléatoire d'intervalles discrets de valeurs.
- *GEN43* - Charge un fichier PVOCEX contenant une analyse VP.

## Routines GEN de Distorsion Linéaire

- *GEN03* - Génère une table de fonction en évaluant un polynôme.
- *GEN13* - Mémoire un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de première espèce.
- *GEN14* - Mémoire un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de seconde espèce.
- *GEN15* - Crée deux tables de fonctions polynomiales mémorisées.

## Routines GEN de Dimensionnement de l'Amplitude

- *GEN04* - Génère une fonction de normalisation.
- *GEN12* - Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée.
- *GEN24* - Lit les valeurs numériques d'une table de fonction déjà allouée en les reproportionnant.

## Routines GEN de Mixage

- *GEN18* - Écrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes.
- *GEN31* - Mélange n'importe quelle forme d'onde définie dans une table existante.
- *GEN32* - Mélange n'importe quelle forme d'onde, rééchantillonnée soit par TFR soit par interpolation linéaire.

## Routines GEN de Hauteur et d'Accordage

- *GEN51* - Remplit une table avec une échelle micro-tonale entièrement personnalisée, à la manière des

opcodes *cpstun*, *cpstuni* et *cpstmid*.

## Routines GEN Nommées

On peut ajouter des routines GEN à Csound au moyen de plugins de fonction GEN. Il y a actuellement un seul plugin GEN qui fournit les fonctions exponentielle et tangente hyperbolique. Ces fonctions GEN ne sont pas appelées par un numéro, mais par un nom.

- "tanh" - remplit une table à partir d'une formule de tangente hyperbolique.
- "exp" - remplit une table à partir d'une formule de tangente hyperbolique.

# GEN01

GEN01 — Transfère des données d'un fichier son dans une table de fonction.

## Description

Ce sous-programme transfère des données d'un fichier son dans une table de fonction.

## Syntaxe

```
f#  date  taille  1  codfic  decal  format  canal
```

## Exécution

*taille* -- nombre de points dans la table. Ordinairement une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*) ; la taille de table maximale est de 16777216 ( $2^{24}$ ) points. L'allocation de mémoire pour la table peut être *différée* en mettant ce paramètre à 0 ; la taille allouée est alors le nombre de points dans le fichier (probablement pas une puissance de 2), et la table n'est pas utilisable par les oscillateurs normaux, mais par l'unité *loscil*. Le fichier son peut aussi être mono ou stéréo.

*codfic* -- entier ou chaîne de caractères dénotant le nom du fichier son source. Un entier dénote le fichier *soundin.codfic* ; une chaîne de caractères (entre apostrophes doubles, espaces autorisés) donne le nom du fichier lui-même, optionnellement un nom de chemin complet. Si le chemin n'est pas complet, le fichier est d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) enfin par SFDIR. Voir aussi *soundin*.

*decal* -- commence à lire à *decal* secondes dans le fichier.

*canal* -- numéro du canal à lire. 0 indique de lire tous les canaux.

*format* -- spécifie le format des données audio :

1 - 8-bit caractères signés	4 - 16-bit entiers courts
2 - 8-bit octets A-law	5 - 32-bit entiers longs
3 - 8-bit octets U-law	6 - 32-bit flottants

Si *format* = 0 le format des échantillons est lu dans l'en-tête du fichier son ou, par défaut depuis l'option *-o* de la ligne de commande de Csound.



### Note

- La lecture s'arrête à la fin du fichier ou lorsque la table est pleine. Les cellules de la table non remplies contiendront des zéros.
- Si *p4* est positif, la table sera post-normalisée (reproportionnée avec une valeur absolue maximale de 1 après génération). Une valeur de *p4* négative empêche cette opération.

## Exemples

Voici un exemple simple de la routine GEN01. Il utilise les fichiers *gen01.csd* [examples/gen01.csd], et *beats.wav* [examples/beats.wav]. Il utilise le fichier audio "beats.wav" dont voici le graphe :



Graphe de la forme d'onde générée par GEN01.

### Exemple 633. Un exemple simple de la routine GEN01.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen01.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 1
  ifn = 1
  ibas = 1

  ; Play the audio sample stored in Table #1.
  al loscil kamp, kcps, ifn, ibas
  out al
endin

</CsInstruments>
<CsScore>

; Table #1: read an audio file (using GEN01).
f 1 0 131072 1 "beats.wav" 0 4 0

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de la routine GEN01. Csound calculera automatiquement la taille de la table parce que nous l'avons fixée à 0. Cet exemple utilise les fichiers *gen01computed.csd* [examples/gen01computed.csd] et *beats.wav* [examples/beats.wav].

### Exemple 634. Un exemple de la routine GEN01 avec une taille de table calculée.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```



```
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen01computed.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 1
  ifn = 1
  ibas = 1

  ; Play the audio sample stored in Table #1.
  al loscil kamp, kcps, ifn, ibas
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1: an audio file (using GEN01).
; Since our table size is 0, Csound will compute it.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Exemples écrits par Kevin Conder

Décembre 2002. Remerciements à Kanata Motohashi pour la correction des erreurs dans les exemples.

Septembre 2003. Remerciements au Dr. Richard Boulanger pour avoir signalé les références au format de fichier AIFF. GEN01 fonctionne aussi avec des fichiers WAV.

# GEN02

GEN02 — Transfère les données des p-champs dans une table de fonction.

## Description

Ce sous-programme transfère les données des p-champs dans une table de fonction.

## Syntaxe

```
f # date taille 2 v1 v2 v3 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La taille de table maximale est de 16777216 ( $2^{24}$ ) points.

*v1*, *v2*, *v3*, etc. -- valeurs à copier directement dans l'espace de la table. Le nombre de valeurs est limité par la variable de compilation *PMAX*, qui contrôle le nombre maximum de p-champs (actuellement 1000). Les valeurs copiées peuvent comprendre le point de garde de la table ; les cellules de la table non remplies contiendront des zéros.



### Note

Si *p4* (le numéro de la routine GEN) est positif, la table sera post-normalisée (reproportionnée avec une valeur absolue maximale de 1 après génération). Une valeur de *p4* négative empêche cette opération. On utilisera habituellement la valeur -2 avec cette fonction GEN, afin que les valeurs ne soient pas normalisées.

## Exemples

Voici un exemple simple de la routine GEN02. Il utilise le fichier *gen02.csd* [examples/gen02.csd]. Il place 12 valeurs plus une valeur de garde explicite pour lecture cyclique dans une table de taille égale à la puissance de 2 supérieure la plus proche. La normalisation est empêchée. Voici le graphe :



Graphe de la forme d'onde générée par GEN02.

### Exemple 635. Un exemple simple de la routine GEN02.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o gen02.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kamp tablei kndx, ifn, ixmode

; Create a sine wave, use the Table #1 values to control
; the amplitude. This creates a sound with a long attack.
al oscil kamp*30000, 440, 2
out al
endin

</CsInstruments>
<CsScore>

; Table #1: an envelope with a long attack (using GEN02).
f 1 0 16 2 0 1 2 3 4 5 6 7 8 9 10 11 0
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

GEN17

## Crédits

Décembre 2002. Merci à Rasmus Ekman, pour avoir corrigé la limite de la variable *PMAX*.

# GEN03

GEN03 — Génère une table de fonction en évaluant un polynôme.

## Description

Ce sous-programme génère une table de fonction en évaluant un polynôme en x sur un intervalle fixe et avec des coefficients spécifiés.

## Syntaxe

```
f # date taille 3 xval1 xval2 c0 c1 c2 ... cn
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1.

*xval1*, *xval2* -- limites gauche et droite de l'intervalle x sur lequel le polynôme est défini (*xval1* < *xval2*). Celles-ci produiront la 1ère valeur stockée et la (puissance-de-2 plus 1)ème valeur stockée respectivement dans la table de la fonction générée.

*c0*, *c1*, *c2*, ..., *cn* -- coefficients du polynôme d'ordre n

$$C_0 + C_1x + C_2x^2 + \dots + C_nx^n$$

Les coefficients peuvent être des nombres réels positifs ou négatifs ; un zéro dénote un terme manquant dans le polynôme. La liste de coefficients commence en p7, avec une limite maximale actuelle de 144 termes.



### Note

- Le segment défini [fn(*xval1*), fn(*xval2*)] est distribué également. Ainsi une table de 512 points sur l'intervalle [-1,1] aura son origine à la cellule 257 (au début de la seconde moitié). Si le point de garde est requis, les deux valeurs fn(-1) et fn(1) existeront dans la table.
- GEN03* est utile en conjonction avec *table* ou *tablei* pour le waveshaping audio (modification du son par distortion non-linéaire). Les coefficients pour produire un formant particulier à partir d'un index de lecture sinusoïdal d'amplitude connue peuvent être déterminés avant le traitement en utilisant des algorithmes tels que les formules de Tchebychev. Voir aussi *GEN13*.

## Exemples

Voici un exemple simple de la routine GEN03. Il utilise le fichier *gen03.csd* [examples/gen03.csd]. Il remplit une table avec une fonction polynomiale du 4ème ordre sur l'intervalle des x allant de -1 à 1. L'origine sera à la position décalée 512. La fonction est post-normalisée. Voici le graphe :



Graphe de la forme d'onde générée par GEN03.

### Exemple 636. Un exemple simple de la routine GEN03.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o gen03.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kamp table kndx, ifn, ixmode

; Create a sine wave, use the Table #1 values to control
; the amplitude.
a1 oscil kamp*30000, 440, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a polynomial function (using GEN03).
f 1 0 1025 3 -1 1 5 4 3 2 2 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

GEN13, GEN14 et GEN15.

# GEN04

GEN04 — Génère une fonction de normalisation.

## Description

Ce sous-programme génère une fonction de normalisation en examinant le contenu d'une table existante.

## Syntaxe

```
f # temps taille 4 source# modesource
```

## Initialisation

*taille* -- nombre de points dans la table. Une puissance-de-2 plus 1. Ne doit pas dépasser (sauf de 1) la taille de la table source examinée ; limitée à exactement la moitié de cette taille si *modesource* est de type décalage (voir ci-dessous).

*source #* -- numéro de table de la fonction stockée à examiner.

*modesource* -- une valeur codée, spécifiant comment la table source doit être parcourue pour obtenir la fonction de normalisation. Zéro indique que la source doit être parcourue de gauche à droite. Une valeur non nulle indique que la source a une structure bipolaire ; la lecture commencera au point médian et progressera vers les extrémités, par paires de points équidistants du centre.



### Note

- La fonction de normalisation dérive de la progression des maxima absolus de la table source parcourue. La nouvelle table est créée de gauche à droite, en stockant des valeurs égales à  $1/(\text{maximum absolu lu jusqu'à là})$ . Les valeurs stockées commenceront ainsi par  $1/(\text{première valeur lue})$ , et deviendront progressivement plus petites lorsque de nouveaux maxima seront rencontrés. Pour une table source normalisée (valeurs  $\leq 1$ ), les valeurs dérivées descendront de  $1/(\text{première valeur lue})$  jusqu'à 1. Si la première valeur lue est zéro, son inverse sera fixé à 1.
- la fonction de normalisation générée par *GEN04* n'est pas elle-même normalisée.
- *GEN04* est utile pour modifier l'échelle d'un signal dérivé d'une table afin qu'il ait une amplitude de crête consistante. On l'utilise particulièrement en waveshaping quand la porteuse (ou fonction d'indexation) a une amplitude inférieure à la moitié de l'échelle complète.

## Exemples

```
f 2 0 512 4 1 1
```

Création d'une fonction de normalisation à utiliser en connexion avec le table 1 de l'exemple *GEN03*. Un décalage bipolaire à point médian est spécifié.

# GEN05

GEN05 — Construit des fonctions à partir de morceaux de courbes exponentielles.

## Description

Construit des fonctions à partir de morceaux de courbes exponentielles.

## Syntaxe

```
f # date taille 5 a n1 b n2 c ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*a*, *b*, *c*, etc. -- valeurs d'ordonnée, dans les p-champs de numéros impairs p5, p7, p9, . . . Elle doivent être non nulles et de même signe.

*n1*, *n2*, etc. -- longueurs des morceaux (nombre de positions mémorisées), dans les p-champs de numéros pairs. Ne peuvent pas être négatives, mais un zéro est significatif pour spécifier des formes d'onde discontinues (comme dans l'exemple ci-dessous). La somme  $n1 + n2 + \dots$  sera normalement égale à *taille* pour les fonctions complètement spécifiées. Si la somme est inférieure, les positions de la fonction non comprises seront mises à zéro ; si la somme est supérieure, seules les premières *taille* positions seront stockées.



### Note

- Si p4 est positif, les fonctions sont post-normalisées (reproportionnées avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.
- Une interpolation linéaire sur des points discrets implique une augmentation ou une diminution le long d'un segment par des sauts égaux entre des positions adjacentes ; une interpolation exponentielle implique une progression par rapports égaux. Dans les deux formes l'interpolation de *a* à *b* suppose que la valeur *b* sera atteinte à la (n + 1)ème position. Pour les fonctions discontinues, et pour les segments dépassant la dernière position, cette valeur ne sera pas atteinte, bien qu'elle puisse éventuellement apparaître comme résultat d'une mise à l'échelle finale.

## Exemples

Voici un exemple simple de la routine GEN05. Il utilise le fichier *gen05.csd* [examples/gen05.csd]. Il créera une jolie enveloppe d'amplitude percussive. Voici le graphe :





Graphique de la forme d'onde générée par GEN05.

### Exemple 637. Un exemple simple de la routine GEN05.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen05.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kamp table kndx, ifn, ixmode

; Create a sine wave, use the Table #1 values to control
; the amplitude. This creates a nice percussive sound.
a1 oscil kamp*30000, 440, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a percussive envelope (using GEN05).
f 1 0 64 5 1 2 120 60 1 1 0.001 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

GEN06, GEN07 et GEN08

## Crédits

Exemple écrit par Kevin Conder

# GEN06

GEN06 — Génère une fonction composée de morceaux de polynômes cubiques.

## Description

Ce sous-programme génèrera une fonction composée de morceaux de polynômes cubiques, couvrant les points spécifiés trois par trois.

## Syntaxe

```
f # date taille 6 a n1 b n2 c n3 d ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*a, c, e, ...* -- les maxima ou les minima locaux des morceaux successifs, dépendant de la relation de ces points avec les inflexions adjacentes. Peuvent être positifs ou négatifs.

*b, d, f, ...* -- ordonnées des points d'inflexion aux extrémités des segments curvilignes successif. Peuvent être positifs ou négatifs.

*n1, n2, n3 ...* -- nombre de valeurs stockées entre les points spécifiés. Ne peuvent pas être négatifs, mais un zéro est significatif pour spécifier des discontinuités. La somme  $n1 + n2 + \dots$  sera normalement égale à *taille* pour les fonctions complètement spécifiées. (Pour des détails, voir *GEN05*).



### Note

*GEN06* construit une fonction stockée à partir de fonctions polynomiales cubiques. Les morceaux groupent les valeurs d'ordonnée par groupes de 3 : point d'inflexion, maximum/minimum, point d'inflexion. Le premier segment complet comprend *b, c, d* et il a pour longueur  $n2 + n3$ , le suivant comprend *d, e, f* et il a pour longueur  $n4 + n5$ , etc. Le premier morceau (*a, b* de longueur *n1*) est incomplet avec seulement une inflexion ; le dernier morceau peut être incomplet aussi. Bien que les points d'inflexion *b, d, f ...* figurent chacun dans deux segments (un à gauche et un à droite), les pentes des deux segments restent indépendantes à ce point commun (c'est-à-dire que la dérivée première sera probablement discontinue). Quand *a, c, e...* sont alternativement maximum et minimum, les jointures des inflexions seront relativement douces ; pour des maxima successifs ou des minima successifs les inflexions seront en peigne.

## Exemples

Voici un exemple simple de la routine GEN06. Il utilise le fichier *gen06.csd* [examples/gen06.csd]. Il crée une courbe allant de 0 à 1 puis à -1, avec un minimum, un maximum et un minimum à ces valeurs respectives. Les inflexions sont à .5 et 0 et sont relativement douces. Voici son graphe :



Graphique de la forme d'onde générée par GEN06.

### Exemple 638. Un exemple simple de la routine GEN06.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen06.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kval table kndx, ifn, ixmode

; Generate a sine waveform, use our Table #1 value to
; vary its frequency by 100 Hz from its base frequency.
ibasefreq = 440
kfreq = kval * 100
a1 oscil 20000, ibasefreq + kfreq, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a curve (using GEN06).
f 1 0 65 6 0 16 0.5 16 1 16 0 16 -1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

GEN05, GEN07 et GEN08

# GEN07

GEN07 — Construit des fonctions à partir de morceaux de lignes droites.

## Description

Construit des fonctions à partir de morceaux de lignes droites.

## Syntaxe

`f # date taille 7 a n1 b n2 c ...`

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir *instruction f*).

*a, b, c, etc.* -- valeurs d'ordonnée, dans les p-champs de numéros impairs p5, p7, p9, ...

*n1, n2, etc.* -- longueur de segment (nombre de positions en mémoire), dans les p-champs de numéros pairs. Ne peuvent pas être négatifs, mais un zéro est significatif pour spécifier des formes d'onde discontinues (comme dans l'exemple ci-dessous). La somme  $n1 + n2 + \dots$  sera normalement égale à *taille* pour les fonctions complètement spécifiées. Si la somme est inférieure, les positions de la fonction non comprises seront mises à zéro ; si la somme est supérieure, seules les premières *taille* positions seront stockées.



### Note

- Si p4 est positif, les fonctions sont post-normalisées (reproportionnées avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.
- Une interpolation linéaire sur des points discrets implique une augmentation ou une diminution le long d'un segment par des sauts égaux entre des positions adjacentes ; une interpolation exponentielle implique une progression par rapports égaux. Dans les deux formes l'interpolation de *a* à *b* suppose que la valeur *b* sera atteinte à la  $(n + 1)$ ème position. Pour les fonctions discontinues, et pour les segments dépassant la dernière position, cette valeur ne sera pas atteinte, bien qu'elle puisse éventuellement apparaître comme résultat d'une mise à l'échelle finale.

## Exemples

Voici un exemple simple de la routine GEN07. Il utilise le fichier *gen07.csd* [examples/gen07.csd]. Il créera une période d'une onde en dent de scie dont la discontinuité se trouve au milieu de la fonction stockée. Voici le graphe :



Graphe de la forme d'onde générée par GEN07.

### Exemple 639. Un exemple simple de la routine GEN07.

Voir les sections *Audio en Temps-Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen07.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the sine wave stored in Table #1.
  al oscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1: a sawtooth wave (using GEN07).
f 1 0 256 7 0 128 1 0 -1 128 0

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsSoundSynthesizer>
```

## Voir Aussi

GEN05, GEN06 et GEN08

# GEN08

GEN08 — Génère une courbe spline cubique par morceaux.

## Description

Ce sous-programme génèrera une courbe spline cubique par morceaux, la plus lisse possible le long de tous les points spécifiés.

## Syntaxe

```
f # date taille 8 a n1 b n2 c n3 d ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir *instruction f*).

*a*, *b*, *c*, etc. -- valeurs d'ordonnée de la fonction.

*n1*, *n2*, *n3* ... -- longueur de chaque segment mesurée en valeurs mémorisées. Ne peuvent pas être nulles, mais peuvent être fractionnaires. Un segment particulier peut stocker ou non des valeurs ; les valeurs stockées seront générées à des points entiers à partir de début de la fonction. La somme  $n1 + n2 + \dots$  sera normalement égale à *taille* pour les fonctions complètement spécifiées.

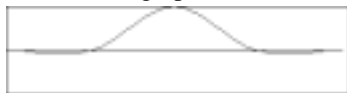


### Note

- *GEN08* construit une table stockée à partir de morceaux d'une fonction polynomiale cubique. Chaque segment s'étend entre deux points spécifiés mais dépend aussi de leurs voisins de chaque côté. Les segments voisins coïncideront en valeur et en pente à leur point commun. (La pente commune est celle d'une parabole passant par ce point et ses deux voisins). La pente aux deux extrémités de la fonction est forcée à zéro (plate).
- *Conseil* : pour créer une discontinuité de pente ou de valeur dans la fonction stockée, disposer une série de points dans l'intervalle entre deux valeurs stockées ; faire de même pour une pente non nulle à l'une des extrémités.

## Exemples

Voici un exemple simple de la routine GEN08. Il utilise le fichier *gen08.csd* [examples/gen08.csd]. Il créera une bosse lisse au milieu, légèrement négative des deux côtés, s'aplatissant ensuite aux extrémités. Voici le graphe :



Graphe de la fonction générée par GEN08.

## Exemple 640. Un exemple simple de la routine GEN08.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen08.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kval table kndx, ifn, ixmode

; Generate a sine waveform, use our Table #1 value to
; vary its frequency by 100 Hz from its base frequency.
ibasefreq = 440
kfreq = kval * 100
a1 oscil 20000, ibasefreq + kfreq, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a curve with a smooth hump (using GEN08).
f 1 0 65 8 0 16 0 16 1 16 0 16 0
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

GEN05, GEN06 et GEN07

# GEN09

GEN09 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

## Description

Ce sous-programme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 3 p-champs avec *GEN09*.

## Syntaxe

```
f # date taille 9 pna ampa phsa pnb ampb phsb ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*pna*, *pnb*, etc. -- numéro de partiel (par rapport à un fondamental qui occuperait *taille* positions par période) des sinus a, sinus b, etc. Doit être positif, mais pas nécessairement un nombre entier, c'est-à-dire que des partiels non harmoniques sont autorisés. Les partiels peuvent être dans n'importe quel ordre.

*ampa*, *ampb*, etc. -- amplitude des partiels *pna*, *pnb*, etc. Ce sont des amplitudes relatives, car la forme d'onde complexe peut être reproporionnée à posteriori. On peut utiliser des valeurs négatives pour signifier une opposition de phase (180 degrés).

*phsa*, *phsb*, etc. -- phase initiale des partiels *pna*, *pnb*, etc., exprimée en degrés (0-360).



### Note

- Ces sous-programmes génèrent des fonctions stockées qui sont la somme de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas, l'onde complexe, une fois calculée, est reproporionnée à l'unité si *p4* est positif. Un *p4* négatif empêchera cette opération.

## Exemples

Voici un exemple simple de la routine *GEN09*. Il utilise le fichier *gen09.csd* [examples/gen09.csd]. Il générera une onde cosinus, une onde sinusoïdale avec une phase initiale de 90 degrés. Voici son graphe :



Graphique de la forme d'onde générée par *GEN09*.



## Exemple 641. Un exemple simple de la routine GEN09.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen09.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the waveform stored in Table #1.
  al oscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1: a cosine wave (using GEN09).
; This is a sine wave with an initial phase of 90 degrees.
f 1 0 16384 9 1 1 90

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de la routine GEN09. Il utilise le fichier *gen09square.csd* [exemples/gen09square.csd]. Il combine les partiels 1, 3 et 9 avec les intensités relatives qu'ils ont dans une onde carrée, sauf que le partiel 9 est inversé. La fonction sera normalisée. Voici le graphe :



Graphe de la forme d'onde générée par GEN09.

## Exemple 642. Une onde carrée générée par la routine GEN09.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o gen09square.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the waveform stored in Table #1.
  al oscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1: an approximation of a square wave (using GEN09).
f 1 0 16384 9 1 3 0 3 1 0 9 0.3333 180

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*GEN10, GEN19*

## Crédits

L'exemple simple a été écrit par Kevin Conder.

# GEN10

GEN10 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

## Description

Ce sous-programme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 1 p-champ avec *GEN10*.

## Syntaxe

```
f # date taille 10 amp1 amp2 amp3 amp4 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*amp1*, *amp2*, *amp3*, etc. -- amplitudes relatives des partiels harmoniques fixes de numéro 1, 2, 3, etc., commençant en p5. Les partiels non désirés recevront une amplitude nulle.



### Note

- Ces sous-programmes génèrent des fonctions stockées qui sont la somme de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas, l'onde complexe, une fois calculée, est reproporionnée à l'unité si p4 est positif. Un p4 négatif empêchera cette opération.

## Exemples

Voici un exemple de la routine GEN10. Il utilise le fichier *gen10.csd* [exemples/gen10.csd]. Il générera une onde sinus simple. Voici son graphe :



Graphe de la forme d'onde générée par GEN10.

### Exemple 643. Un exemple de la routine GEN10.

Voir les sections *Audio en Temps-Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen10.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the sine wave stored in Table #1.
  al oscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave (using GEN10).
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*GEN09*, *GEN11* et *GEN19*.

## Crédits

Exemple écrit par Kevin Conder

# GEN11

GEN11 — Génère un ensemble additif de partiels cosinus.

## Description

Ce sous-programme génère un ensemble additif de partiels cosinus, à la manière des générateurs de Csound *buzz* et *gbuzz*.

## Syntaxe

```
f # date taille lh nh [lh] [r]
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*nh* -- nombre d'harmoniques demandés. Doit être positif.

*lh*(optional) -- harmonique présent le plus bas. Peut être positif, nul ou négatif. L'ensemble d'harmoniques peut démarrer à n'importe quel numéro d'harmonique et progresse vers le haut ; si *lh* est négatif, tous les harmoniques en dessous de zéro se réfléchiront autour de zéro pour produire des harmoniques positifs sans changement de phase (car le cosinus est une fonction paire), et s'ajouteront de façon constructive aux harmoniques positifs de l'ensemble. La valeur par défaut est 1.

*r*(facultatif) -- multiplicateur dans une série de coefficients d'amplitude. C'est une séries de puissances : si le *lh* ème harmonique a un coefficient d'amplitude de *A* le  $(lh + n)$ ème harmonique aura un coefficient de  $A * r^n$ , c'est-à-dire que les valeurs d'amplitudes suivent une courbe exponentielle. *r* peut être positif, nul ou négatif, et n'est pas restreint à des entiers. La valeur par défaut est 1.



### Note

- Ce sous-programme est une version invariante dans le temps des générateurs de Csound *buzz* et *gbuzz*, et il est similairement utile comme source sonore complexe pour la synthèse soustractive. Si *lh* et *r* sont utilisés, il agit comme *gbuzz* ; si les deux sont absents ou égaux à 1, il se réduit au générateur plus simple *buzz* (c'est-à-dire *nh* harmoniques d'amplitude égale commençant avec le fondamental).
- Lire la forme d'onde stockée avec un oscillateur est plus efficace que d'utiliser les unités dynamiques *buzz*. Cependant, le contenu spectral est invariant et il faut faire attention à ce que les harmoniques les plus hauts ne dépassent pas la fréquence de Nyquist pour éviter les repliements.

## Exemples

Voici un exemple simple de la routine GEN11. Il utilise le fichier *gen11.csd* [examples/gen11.csd]. Il génèrera une onde cosinus simple. Voici son graphe :



Graphique de la forme d'onde générée par GEN11.

### Exemple 644. Un exemple simple de la routine GEN11.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen11.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 440
  ifn = 1

  ; Play the cosine wave stored in Table #1.
  al oscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1: a simple cosine wave (using GEN11).
f 1 0 16384 11 1 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*GEN10*

## Crédits

Exemple écrit par Kevin Conder

# GEN12

GEN12 — Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée.

## Description

Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée, d'ordre 0, adaptée pour la MF modulée en amplitude.

## Syntaxe

```
f # date taille 12 intx
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*intx* -- spécifie l'intervalle des  $x$  [0 à +*intx*] sur lequel la fonction est définie.



### Note

- Ce sous-programme calcule le logarithme naturel d'une fonction de Bessel de seconde espèce modifiée, d'ordre 0 (habituellement écrite comme  $I_0$ ), sur l'intervalle des  $x$  demandé. Cet appel devrait désactiver la normalisation.
- Cette fonction est utile comme facteur d'échelle d'amplitude dans la MF à période synchrone modulée en amplitude. (Voir Palamin & Palamin, *J. Audio Eng. Soc.*, 36/9, Sept. 1988, pp.671-684.) L'algorithme est intéressant car il permet de rendre le spectre de MF, habituellement symétrique, assymétrique autour d'une fréquence autre que la porteuse, et il est ainsi utile pour placer des formants. En utilisant un index de lecture dans la table de  $I(r - 1/r)$ , où  $I$  est l'index de modulation et  $r$  est un paramètre exponentiel affectant l'importance des partiels, l'algorithme Palamin se montre relativement efficace, ne demandant que des oscil, des lecture de table, et un appel d'*exp*.

## Exemples

Voici un exemple simple de la routine GEN12. Il utilise le fichier *gen12.csd* [examples/gen12.csd]. Il génère la fonction  $\ln(I_0(x))$  de 0 à 20. Voici son graphe :



Graphe de la forme d'onde générée par GEN12.

**Exemple 645. Un exemple simple de la routine GEN12.**

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen12.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kamp tablei kndx, ifn, ixmode

; Create a sine wave, use the Table #1 values to control
; the amplitude. This creates a sound with a long attack.
a1 oscil kamp*30000, 440, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a modified Bessel function (using GEN12).
f 1 0 2049 12 20
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Exemple écrit par Kevin Conder



# GEN13

GEN13 — Mémorise un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de première espèce.

## Description

Utilise les coefficients de Tchebychev pour générer des fonctions polynomiales stockées qui, dans le waveshaping, peuvent être utilisées pour séparer une sinus en harmoniques selon un spectre prédéfini.

## Syntaxe

```
f # date taille 13 xint xamp h0 h1 h2 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.

*xint* -- fournit les valeurs gauches et droites *[-xint, +xint]* de l'intervalle des *x* sur lequel le polynôme doit être évalué. *GEN13* et *GEN14* appellent *GEN03* pour évaluer leurs fonctions ; la valeur en *p5* est ainsi étendue en une paire négative-positive *p5*, *p6* avant l'appel de *GEN03*. La valeur normale est 1.

*xamp* -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

*h0*, *h1*, *h2*, etc. -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

$xamp * \text{int}(\text{taille}/2)/xint$

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.



### Note

*GEN13* est le générateur de fonction normalement employé dans le waveshaping standard. Il stocke un polynôme dont les coefficients dérivent des polynômes de Tchebychev de première espèce, de sorte qu'une sinus d'amplitude *xamp* pilotant le dispositif produise le spectre spécifié en sortie. Noter que l'évolution de ce spectre ne varie généralement pas linéairement en fonction de *xamp*. Cependant, il est à bande limitée (les seuls harmoniques qui apparaissent seront ceux qui auront été spécifiés au moment de la génération) ; et les harmoniques auront tendance à apparaître et à se développer en ordre ascendant (les harmoniques inférieurs dominant pour de faibles *xamp*, et la richesse spectrale augmentant pour des valeurs plus grandes de *xamp*). Une valeur *hn* négative implique une opposition de phase de cet harmonique ; le spectre d'amplitude complet demandé ne sera pas affecté par ce déphasage, bien que l'évolution de plusieurs de ses harmoniques puisse l'être. Le schéma +, +, -, -, +, +, ... pour *h0*, *h1*, *h2*, ... minimisera le problème de la normalisation pour de faibles valeurs de *xamp* (voir ci-dessus), mais ne fournira pas nécessairement le schéma d'évolution le plus lisse.

## Exemples

Voici un exemple simple de la routine GEN13. Il utilise le fichier *gen13.csd* [examples/gen13.csd]. Il crée une fonction qui, lors du waveshaping, séparera une sinus en 3 harmoniques impairs d'importance relative 5:3:1. Voici son graphe :



Graphe de la forme d'onde générée par GEN13.

### Exemple 646. Un exemple simple de la routine GEN13.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen13.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kval table kndx, ifn, ixmode

; Generate a sine waveform, use our Table #1 value to
; vary its frequency by 100 Hz from its base frequency.
ibasefreq = 440
kfreq = kval * 100
al oscil 20000, ibasefreq + kfreq, 2
out al
endin

</CsInstruments>
<CsScore>

; Table #1: a polynomial function (using GEN13).
f 1 0 1025 13 1 1 0 5 0 3 0 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

*GEN03*, *GEN14* et *GEN15*.

# GEN14

GEN14 — Mémoire un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de seconde espèce.

## Description

Utilise les coefficients de Tchebychev pour générer des fonctions polynomiales stockées qui, dans le waveshaping, peuvent être utilisées pour séparer une sinus en harmoniques selon un spectre prédéfini.

## Syntaxe

```
f # date taille 14 xint xamp h0 h1 h2 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.

*xint* -- fournit les valeurs gauches et droites  $[-xint, +xint]$  de l'intervalle des  $x$  sur lequel le polynôme doit être évalué. *GEN13* et *GEN14* appellent *GEN03* pour évaluer leurs fonctions ; la valeur en *p5* est ainsi étendue en une paire négative-positive *p5*, *p6* avant l'appel de *GEN03*. La valeur normale est 1.

*xamp* -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

*h0*, *h1*, *h2*, etc. -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

$xamp * \text{int}(\text{taille}/2)/xint$

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.



### Note

- *GEN13* est le générateur de fonction normalement employé dans le waveshaping standard. Il stocke un polynôme dont les coefficients dérivent des polynômes de Tchebychev de première espèce, de sorte qu'une sinus d'amplitude *xamp* pilotant le dispositif produise le spectre spécifié en sortie. Noter que l'évolution de ce spectre ne varie généralement pas linéairement en fonction de *xamp*. Cependant, il est à bande limitée (les seuls harmoniques qui apparaissent seront ceux qui auront été spécifiés au moment de la génération) ; et les harmoniques auront tendance à apparaître et à se développer en ordre ascendant (les harmoniques inférieurs dominant pour de faibles *xamp*, et la richesse spectrale augmentant pour des valeurs plus grandes de *xamp*). Une valeur *hn* négative implique une opposition de phase de cet harmonique ; le spectre d'amplitude complet demandé ne sera pas affecté par ce déphasage, bien que l'évolution de plusieurs de ses harmoniques puisse l'être. Le schéma +, +, -, -, +, +, ... pour *h0*, *h1*, *h2*, ... minimisera le problème de la normalisation pour de faibles valeurs de *xamp* (voir ci-dessus), mais ne fournira pas nécessairement le schéma d'évolution le plus lisse.

- *GEN14* stocke un polynôme dont les coefficients dérivent de polynômes de Tchebychev de seconde espèce.

## Exemples

Voici un exemple simple de la routine GEN14. Il utilise le fichier *gen14.csd* [examples/gen14.csd]. Il crée une fonction qui, lors du waveshaping, séparera une sinus en 3 harmoniques impairs d'importance relative 5:3:1. Voici son graphe :



Graphe de la forme d'onde générée par GEN14.

### Exemple 647. Un exemple simple de la routine GEN14.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o gen14.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kval table kndx, ifn, ixmode

; Generate a sine waveform, use our Table #1 value to
; vary its frequency by 100 Hz from its base frequency.
ibasefreq = 440
kfreq = kval * 100
al oscil 20000, ibasefreq + kfreq, 2
out al
endin

</CsInstruments>
<CsScore>

; Table #1: a polynomial function (using GEN14).
f 1 0 1025 14 1 1 0 5 0 3 0 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
```

e

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir Aussi

*GEN03*, *GEN13* et *GEN15*.

## Crédits

Exemple écrit par Kevin Conder

# GEN15

GEN15 — Crée deux tables de fonctions polynomiales mémorisées.

## Description

Ce sous-programme crée deux tables de fonctions polynomiales mémorisées, appropriées pour une utilisation en quadrature de phase.

## Syntaxe

```
f # date taille 15 xint xamp h0 phs0 h1 phs1 h2 phs2 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.

*xint* -- fournit les valeurs gauches et droites  $[-xint, +xint]$  de l'intervalle des  $x$  sur lequel le polynôme doit être évalué. Ce sous-programme appellera éventuellement *GEN03* pour évaluer les deux fonctions ; la valeur en  $p5$  est alors étendue en une paire négative-positive  $p5, p6$  avant l'appel de *GEN03*. La valeur normale est 1.

*xamp* -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

*h0, h1, h2, ..., hn* -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

$xamp * \text{int}(\text{taille}/2)/xint$

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.

*phs0, phs1, ...* -- phase en degrés des harmoniques désirés *h0, h1, ...* lorsque les deux fonctions de *GEN15* sont utilisées en quadrature de phase.



### Note

*GEN15* crée deux tables de même taille, étiquetées  $f\#$  et  $f\# + 1$ . La table  $\#$  contiendra une fonction de Tchebychev de première espèce, évaluée par *GEN03* avec des harmoniques d'amplitude  $h0\cos(phs0), h1\cos(phs1), \dots$ . Table  $\# + 1$  contiendra une fonction de Tchebychev de deuxième espèce, évaluée par *GEN14* avec les harmoniques  $h1\sin(phs1), h2\sin(phs2), \dots$  (noter le déplacement harmonique). Les deux tables peuvent être utilisées en conjonction dans un réseau de waveshaping qui exploite la quadrature de phase.

## Voir Aussi

*GEN03, GEN13* et *GEN14*.

# GEN16

GEN16 — Crée une table depuis une valeur initiale jusqu'à une valeur terminale.

## Description

Crée une table depuis la valeur *deb* jusqu'à la valeur *fin* en *dur* pas.

## Syntaxe

```
f # date taille 16 deb dur type fin
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.

*deb* -- valeur de départ

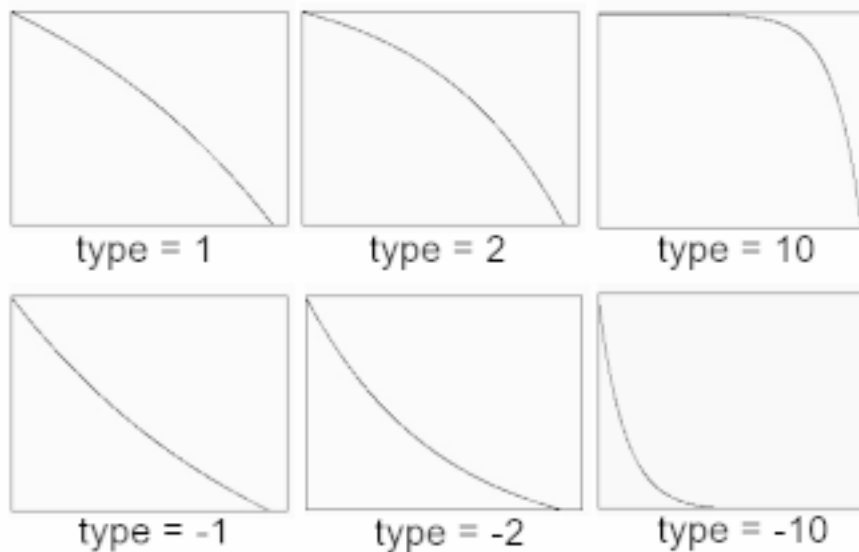
*dur* -- nombre de segments

*type* -- si 0, une ligne droite est produite. Si différent de zéro, alors *GEN16* crée la courbe suivante sur *dur* pas :

$$\text{deb} + (\text{fin} - \text{deb}) * (1 - \exp(i * \text{type} / (\text{dur} - 1))) / (1 - \exp(\text{type}))$$

*fin* -- valeur après *dur* segments

Voici quelques exemples de courbes générées pour différentes valeurs de *type* :



Tables générées par GEN16 pour différentes valeurs de *type*.





## Note

Si  $type > 0$ , on a une courbe montant lentement (concave) ou décroissant lentement (convexe), tandis que si  $type < 0$ , la courbe monte rapidement (convexe) ou décroît rapidement (concave). Voir aussi *transeg*.

### Exemple 648. Un exemple simple de la routine GEN16.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen16.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1

instr 1
  kcps init 1/p3
  kndx phasor kcps

  ifn = p4
  ixmode = 1
  kval table kndx, ifn, ixmode

  ibasefreq = 440
  kfreq = kval * ibasefreq
  a1 oscil 20000, ibasefreq + kfreq, 1
  out a1
endin

</CsInstruments>
<CsScore>

f 1 0 16384 10 1

f 2 0 1024 16 1 1024 1 0
f 3 0 1024 16 1 1024 2 0
f 4 0 1024 16 1 1024 10 0
f 5 0 1024 16 1 1024 -1 0
f 6 0 1024 16 1 1024 -2 0
f 7 0 1024 16 1 1024 -10 0

i 1 0 2 2
i 1 + . 3
i 1 + . 4
i 1 + . 5
i 1 + . 6
i 1 + . 7

e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitich

University of Bath, Codemist. Ltd.  
Bath, UK  
Octobre 2000

Nouveau dans la version 4.09 de Csound

# GEN17

GEN17 — Crée une fonction en escalier à partir des paires x-y données.

## Description

Ce sous-programme crée une fonction en escalier à partir des paires x-y données.

## Syntaxe

```
f # date taille 17 x1 a x2 b x3 c ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.

*x1*, *x2*, *x3*, etc. -- valeurs d'abscisse x, en ordre ascendant, commençant par 0.

*a*, *b*, *c*, etc. -- valeurs y à ces valeurs d'abscisse x, maintenues jusqu'à la valeur d'abscisse x suivante.



### Note

Ce sous-programme crée une fonction en escalier de paires x-y dont les valeurs y sont maintenues vers la droite. La valeur de y la plus à droite est ensuite maintenue jusqu'à la fin de la table. Cette fonction est utile pour mettre en correspondance un ensemble de données avec un autre, tel que des numéros de notes MIDI avec des numéros de tables de sons échantillonnés. (voir *loscil*).

## Exemples

```
f 1 0 128 -17 0 1 12 2 24 3 36 4 48 5 60 6 72 7 84 8
```

Ceci décrit une fonction en escalier avec huit niveaux croissants successifs, chacun occupant 12 positions sauf pour le dernier qui étend sa valeur jusqu'à la fin de la table. La normalisation est empêchée. En indexant cette table avec un numéro de note MIDI, on retrouvera une valeur différente pour chaque octave jusqu'à la huitième, au-delà de laquelle la valeur retournée restera la même.

## Voir Aussi

GEN02

# GEN18

GEN18 — Ecrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes.

## Description

Ecrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes. Chaque forme d'onde utilisée nécessite 4 p-champs et peut se chevaucher avec les autres formes d'onde.

## Syntaxe

```
f # date taille 18 fna ampa debuta fina fnb ampb debutb finb ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance-de-2 plus 1 (voir l'instruction f).

*fna, fnb, etc.* -- numéros des tables pré-existantes à écrire dans la table.

*ampa, ampb, etc.* -- amplitude des formes d'onde. Ces amplitudes sont relatives, car la forme d'onde composée pourra être post-normalisée. Des valeurs négatives sont autorisées et impliquent une opposition de phase.

*debuta, debutb, etc.* -- où commencer à écrire fn dans la table.

*fina, finb, etc.* -- où terminer l'écriture de fn dans la table.

## Exemples

```
f 1 0 4096 10 1
f 2 0 1025 18 1 1 0 512 1 1 513 1025
```

f2 consiste en deux copies de f1 écrites dans les positions 0-512 et 513-1025.

## Noms anciennement utilisés

GEN18 était appelé GEN22 dans la version 4.18. Le nom fut changé à cause d'un conflit avec DirectC-sound.

## Crédits

Auteur : William « Pete » Moss  
University of Texas at Austin  
Austin, Texas USA  
Janvier 2002

Nouveau dans la version 4.18, changé dans la version 4.19

# GEN19

GEN19 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

## Description

Ce sous-programme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 4 p-champs dans *GEN19*.

## Syntaxe

```
f # date taille 19 pna ampa phsa dcoa pnb ampb phsb dcob ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction f).

*pna*, *pnb*, etc. -- numéro de partiel (relativement à un fondamental qui occuperait *taille* positions par période) de sinus a, sinus b, etc. Doit être positif, mais pas nécessairement un nombre entier, c'est-à-dire que des partiels non harmoniques sont autorisés. Les partiels peuvent être dans n'importe quel ordre.

*ampa*, *ampb*, etc. -- amplitude des partiels *pna*, *pnb*, etc. Ces amplitudes sont relatives, car la forme d'onde composée peut être normalisée plus tard. Des valeurs négatives sont autorisées et impliquent une opposition de phase.

*phsa*, *phsb*, etc. -- phase initiale des partiels *pna*, *pnb*, etc., exprimée en degrés.

*dcoa*, *dcob*, etc. -- Décalage CC (Composante Continue) des partiels *pna*, *pnb*, etc. Il est appliqué *après* l'amplitude, c'est-à-dire qu'une valeur de 2 montera une sinus d'amplitude 2 de l'intervalle [-2,2] à l'intervalle [0,4] (avant la normalisation finale).



### Note

- Ces sous-programmes génèrent des fonctions stockées comme sommes de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas l'onde composée, une fois évaluée, est ensuite normalisée à l'unité si p4 est positif. Un p4 négatif empêchera cette opération.

## Exemples

Voici un exemple simple de la routine GEN19. Il utilise le fichier *gen19.csd* [examples/gen19.csd]. Il génèrera une jolie courbe en cloche, voici son graphe :



Graphique de la forme d'onde générée par GEN19.

## Exemple 649. Un exemple simple de la routine GEN19.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o gen19.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create an index over the length of our entire note.
kcps init 1/p3
kndx phasor kcps

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kval table kndx, ifn, ixmode

; Generate a sine waveform, use our Table #1 value to
; vary its frequency by 100 Hz from its base frequency.
ibasefreq = 440
kfreq = kval * 100
a1 oscil 20000, ibasefreq + kfreq, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a bell curve (using GEN19).
f 1 0 16384 -19 1 1 260 1
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 3 seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

## Voir Aussi

GEN09 et GEN10

## Crédits

Exemple écrit par Kevin Conder

# GEN20

GEN20 — Génère les fonctions de différentes fenêtres.

## Description

Ce sous-programme génère les fonctions de différentes fenêtres. Ces fenêtres sont utilisées habituellement pour l'analyse spectrale ou pour des enveloppes de grain.

## Syntaxe

```
f # date taille 20 fenêtre max [opt]
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ( + 1).

*fenêtre* -- Type de la fenêtre à générer :

- 1 = Hamming
- 2 = Hanning
- 3 = Bartlett (triangle)
- 4 = Blackman (3-termes)
- 5 = Blackman - Harris (4-termes)
- 6 = Gaussienne
- 7 = Kaiser
- 8 = Rectangle
- 9 = Sync

*max* -- Pour p4 négatif ce sera la valeur absolue au pic de la fenêtre. Si p4 est positif ou si p4 est négatif et p6 est absent la table sera post-normalisée à une valeur maximale de 1.

*opt* -- Argument facultatif nécessaire pour la fenêtre gaussienne et pour la fenêtre de Kaiser.

## Exemples

```
f 1 0 1024 20 5
```

Crée une fonction qui contient une fenêtre de Blackman - Harris à 4 termes avec une valeur maximale de 1.

**f**      1      0      1024      -20      2      456

Crée une fonction qui contient une fenêtre de Hanning avec une valeur maximale de 456.

**f**      1      0      1024      -20      1

Crée une fonction qui contient une fenêtre de Hamming avec une valeur maximale de 1.

**f**      1      0      1024      20      7      1      2

Crée une fonction qui contient une fenêtre de Kaiser avec une valeur maximale de 1. L'argument supplémentaire spécifie comment la fenêtre est "ouverte", par exemple une valeur de 0 donne une fenêtre rectangulaire et une valeur de 10 donne une fenêtre semblable à une fenêtre de Hamming.

**f**      1      0      1024      20      6      1      2

Crée une fonction qui contient une fenêtre gaussienne avec une valeur maximale de 1. L'argument supplémentaire spécifie la largeur de la fenêtre, comme l'écart type de la courbe ; dans cet exemple l'écart type vaut 2. La valeur par défaut est 1.

Pour les graphes, voir les *Fonctions Fenêtre*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.2 de Csound

L'argument facultatif de la gaussienne a été ajouté dans la version 5.10



# GEN21

GEN21 — Génère les tables de différentes distributions aléatoires.

## Description

Génère les tables de différentes distributions aléatoires. (Voir aussi *betarand*, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand* et *weibull*)

## Syntaxe

```
f # date taille 21 type niveau [arg1 [arg2]]
```

## Initialisation

*date* et *taille* sont les arguments habituels des fonctions GEN. *niveau* définit l'amplitude. Noter que GEN21 n'effectue pas d'auto-normalisation comme le font la plupart des autres fonctions GEN. *type* définit la distribution à utiliser :

- 1 = Uniforme (seulement des nombres positifs)
- 2 = Linéaire (seulement des nombres positifs)
- 3 = Triangulaire (nombres positifs et négatifs)
- 4 = Exponentielle (seulement des nombres positifs)
- 5 = Biexponentielle (nombres positifs et négatifs)
- 6 = Gaussienne (nombres positifs et négatifs)
- 7 = Cauchy (nombres positifs et négatifs)
- 8 = Cauchy Positive (seulement des nombres positifs)
- 9 = Beta (seulement des nombres positifs)
- 10 = Weibull (seulement des nombres positifs)
- 11 = Poisson (seulement des nombres positifs)

De tous ces cas seulement le 9 (Beta) et le 10 (Weibull) ont besoin d'arguments supplémentaires. Beta nécessite deux arguments et Weibull un.

## Exemples

```
f1 0 1024 21 1 ; Uniforme (bruit blanc)
f1 0 1024 21 6 ; Gaussienne
f1 0 1024 21 9 1 1 2 ; Beta (noter que le niveau précède les arguments)
f1 0 1024 21 10 1 2 ; Weibull
```

Toutes les additions ci-dessus furent conçus par l'auteur entre mai et décembre 1994, sous la supervision du Dr Richard Boulanger.

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.2 de Csound

# GEN22

GEN22 — Obsolète.

## Description

Obsolète depuis la version 4.19. Utiliser plutôt la routine *GEN18*.

# GEN23

GEN23 — Lit des valeurs numériques à partir d'un fichier texte.

## Description

Ce sous-programme lit des valeurs numériques à partir d'un fichier ASCII.

## Syntaxe

```
f # date taille -23 "nomfichier.txt"
```

## Initialisation

*"nomfichier.txt"* -- les valeurs numériques contenues dans "nomfichier.txt" (qui indique le nom de chemin complet du fichier de caractères à lire) peuvent être séparées par des espaces, des tabulations, des caractères de passage à la ligne ou des virgules. De plus, on peut utiliser comme commentaires des mots qui contiennent des caractères non numériques car ils sont ignorés.

*taille* -- nombre de points dans la table. Doit être une puissance de 2, une puissance de 2 + 1, ou zéro. Si *taille* = 0, la taille de la table est déterminée par le nombre de valeurs numériques dans *nomfichier.txt*. (Nouveau dans la version 3.57 de Csound)



### Note

Tous les caractères suivant un ';' ou un '#' (commentaire) sont ignorés jusqu'à la ligne suivante (les nombres aussi).

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Février 1998

Nouveau dans la version 3.47 de Csound. Les commentaires commençant par un '#' sont ignorés depuis la version 5.12 de csound.

# GEN24

GEN24 — Lit les valeurs numériques d'une table de fonction déjà allouée en les reproportionnant.

## Description

Ce sous-programme lit les valeurs numériques d'une table de fonction déjà allouée et les reproportionne selon les valeurs *min* et *max* données par l'utilisateur.

## Syntaxe

```
f # date taille -24 ftable min max
```

## Initialisation

*#, date, taille* -- les paramètres GEN habituels. Voir l'instruction *f*.

*ftable* -- *ftable* doit être une table déjà allouée avec la même taille que cette fonction.

*min, max* -- l'intervalle de recadrage.



### Note

Ce GEN est utile, par exemple, pour éliminer le décalage du début dans les morceaux d'exponentielle permettant d'avoir une vrai origine à zéro.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16 de Csound

# GEN25

GEN25 — Construit des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).

## Description

Ces sous-programmes sont utilisés pour construire des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).

## Syntaxe

```
f # date taille 25 x1 y1 x2 y2 x3 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*).

*x1*, *x2*, *x3*, etc. -- positions dans la table auxquelles la valeur *y* suivante devra être atteinte. Doivent être en ordre croissant. Si la dernière valeur est inférieure à la *taille*, les positions restantes seront mises à zéro. Ne doivent pas être négatives mais peuvent être nulles.

*y1*, *y2*, *y3*, etc. -- Valeurs charnière atteintes à la position spécifiée par la valeur *x* précédente. Elles doivent être non nulles et toutes du même signe.



### Note

Si *p4* est positif, les fonctions sont post-normalisées (reproportionnées à une valeur absolue maximale de 1 après génération). Un *p4* négatif empêchera cette opération.

## Voir Aussi

*Instruction f*, GEN27

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.49 de Csound

# GEN27

GEN27 — Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

## Description

Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

## Syntaxe

```
f # date taille 27 x1 y1 x2 y2 x3 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1. (voir l'instruction *f*).

*x1*, *x2*, *x3*, etc. -- positions dans la table auxquelles la valeur *y* suivante devra être atteinte. Doivent être en ordre croissant. Si la dernière valeur est inférieure à la taille, les positions restantes seront mises à zéro. Ne doivent pas être négatives mais peuvent être nulles.

*y1*, *y2*, *y3*, etc. -- Valeurs charnière atteintes à la position spécifiée par la valeur *x* précédente.



### Note

Si *p4* est positif, les fonctions sont post-normalisées (reproportionnées à une valeur absolue maximale de 1 après génération). Un *p4* négatif empêchera cette opération.

## Exemples

```
f 1 0 257 27 0 0 100 1 200 -1 256 0
```

Décrit une fonction qui commence à 0, monte jusqu'à 1 à la 100ème position de la table, descend à -1, à la 200ème position, et revient à 0 à la fin de la table. L'interpolation est linéaire.

## Voir Aussi

*Instruction f*, *GEN25*

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.49 de Csound

# GEN28

GEN28 — Lit un fichier texte qui contient une trajectoire paramétrée par le temps.

## Description

Ce générateur de fonction lit un fichier texte qui contient des ensembles de trois valeurs représentant des coordonnées xy et un paramètre temporel indiquant quand placer le signal à cette position, permettant à l'utilisateur de définir une trajectoire paramétrée par le temps. Le format du fichier est de la forme :

```
temps1  X1  Y1
temps2  X2  Y2
temps3  X3  Y3
```

La configuration des coordonnées xy dans l'espace place le signal de la manière suivante :

- a1 est -1, 1
- a2 est 1, 1
- a3 est -1, -1
- a4 est 1, -1

Cela suppose des haut-parleurs disposés avec a1 en avant gauche, a2 en avant droite, a3 en arrière gauche, a4 en arrière droite. Les valeurs supérieures à 1 provoqueront une atténuation des sons comme s'ils étaient distants. *GEN28* crée les valeurs avec une résolution de 10 millisecondes.

## Syntaxe

```
f # date taille 28 codfic
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être 0. *GEN28* prend une taille de 0 et alloue la mémoire automatiquement.

*codfic* -- chaîne de caractères dénotant le nom du fichier source. Une chaîne de caractères (entre apostrophes doubles, espaces autorisés) donne le nom du fichier lui-même, optionnellement un nom de chemin complet. Si le chemin n'est pas complet, le fichier nommé est cherché dans le répertoire courant.

## Exemples

```
f1 0 0 28 "move"
```



Le fichier "move" ressemblera à ceci :

0	-1	1
1	1	1
2	4	4
2.1	-4	-4
3	10	-10
5	-40	0

Puisque *GEN28* crée les valeurs avec une résolution de 10 millisecondes, il y aura 500 valeurs créées en interpolant entre X1 et X2, X2 et X3, etc., et entre Y1 et Y2, Y2 et Y3, etc., sur le nombre approprié de valeurs qui sont stockées dans la table de fonction. Le son démarrera à l'avant gauche, il bougera pendant 1 seconde vers l'avant droite, durant la seconde suivante il s'éloignera mais toujours à l'avant droite, ensuite il bougera vers l'arrière gauche en seulement 1/10 de seconde, un peu éloigné. Enfin, pendant les 0,9 secondes restantes le son bougera vers l'arrière droite, modérément éloigné, et il viendra s'arrêter entre les deux canaux gauche (plein ouest !), assez éloigné.

## Crédits

Auteur : Richard Karpen  
Seattle, Wash  
1998

Nouveau dans la version 3.48 de Csound

# GEN30

GEN30 — Génère des partiels harmoniques en analysant une table existante.

## Description

Extrait un sous-ensemble de la série harmonique d'une forme d'onde existante.

## Syntaxe

```
f # date taille 30 src minh maxh [ref_sr] [interp]
```

## Exécution

*src* -- ftable source

*minh* -- numéro de l'harmonique le plus bas

*maxh* -- numéro de l'harmonique le plus haut

*ref\_sr* (facultatif) -- *maxh* est pondéré par (*sr* / *ref\_sr*). La valeur par défaut de *ref\_sr* est *sr*. Si *ref\_sr* est nul ou négatif, il est ignoré.

*interp* (facultatif) -- si différent de zéro, permet de changer l'amplitude des harmoniques le plus bas et le plus haut en fonction de la partie fractionnaire de *minh* et *maxh*. Par exemple, si *maxh* vaut 11.3 alors le 12ème harmonique est ajouté avec une amplitude de 0.3. Ce paramètre vaut zéro par défaut.

*GEN30* ne supporte pas les tables avec un point de garde (c'est-à-dire une taille de table = puissance-de-deux + 1). Bien que de telles tables fonctionnent aussi bien en entrée qu'en sortie, lors de la lecture d'une table source, le point de garde est ignoré, et lors de l'écriture de la table en sortie, le point de garde est simplement copié du premier échantillon (index de table = 0).

La raison de cette limitation est que *GEN30* utilise la TFR, qui nécessite que la taille de table soit une puissance de deux. *GEN32* permet l'utilisation de l'interpolation linéaire pour le rééchantillonnage et le déphasage, ce qui rend possible l'utilisation de n'importe quelle taille de table (cependant, pour les partiels calculés par TFR, la limitation de la puissance de deux existe toujours).

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.16

# GEN31

GEN31 — Mélange n'importe quelle forme d'onde définie dans une table existante.

## Description

Cette routine est semblable à GEN09, mais permet le mélange de n'importe quelle forme d'onde définie dans une table existante.

## Syntaxe

```
f # date taille 31 src pna ampa phsa pnb ampb phsb ...
```

## Exécution

*src* -- numéro de la table source

*pna*, *pnb*, ... -- numéro de partiel, doit être un entier positif

*ampa*, *ampb*, ... -- échelle d'amplitude

*phsa*, *phsb*, ... -- phase initiale (0 à 1)

*GEN31* ne supporte pas les tables avec un point de garde (c'est-à-dire une taille de table = puissance-de-deux + 1). Bien que de telles tables fonctionnent aussi bien en entrée qu'en sortie, lors de la lecture d'une table source, le point de garde est ignoré, et lors de l'écriture de la table en sortie, le point de garde est simplement copié du premier échantillon (index de table = 0).

La raison de cette limitation est que *GEN31* utilise la TFR, qui nécessite que la taille de table soit une puissance de deux. *GEN32* permet l'utilisation de l'interpolation linéaire pour le rééchantillonnage et le déphasage, ce qui rend possible l'utilisation de n'importe quelle taille de table (cependant, pour les partiels calculés par TFR, la limitation de la puissance de deux existe toujours).

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

# GEN32

GEN32 — Mélange n'importe quelle forme d'onde, rééchantillonnée soit par TFR soit par interpolation linéaire.

## Description

Cette routine est semblable à *GEN31*, mais elle permet la spécification d'une table source pour chaque partiel. Les tables peuvent être rééchantillonnées soit par TFR soit par interpolation linéaire.

## Syntaxe

```
f # date taille 32 srca pna ampa phsa srcb pnb ampb phsb ...
```

## Exécution

*srca, srcb* -- numéro de table source. Une valeur négative peut être utilisée pour lire une table avec interpolation linéaire (par défaut, la forme d'onde source est transposée et déphasée par TFR) ; c'est moins précis, mais plus rapide, et cela permet des numéros de partiels non entiers et négatifs.

*pná, pnb, ...* -- numéro de partiel, doit être un entier positif si le numéro de la table source est positif (c'est-à-dire rééchantillonnage par TFR).

*ampa, ampb, ...* -- échelle d'amplitude

*phsa, phsb, ...* -- phase initiale (0 à 1)

## Exemples

```
itmp    ftgen 1, 0, 16384, 7, 1, 16384, -1      ; dent de scie
itmp    ftgen 2, 0, 8192, 10, 1                ; sinus
; mélange les tables
itmp    ftgen 5, 0, 4096, -32, -2, 1.5, 1.0, 0.25, 1, 2, 0.5, 0, \
                                     1, 3, -0.25, 0.5
; fenêtre
itmp    ftgen 6, 0, 16384, 20, 3, 1
; génère des formes d'onde à bande limitée
inote   = 0
loop0:
icps    = 440 * exp(log(2) * (inote - 69) / 12)      ; une table pour
inumh   = sr / (2 * icps)                             ; chaque numéro de note MIDI
ift     = int(inote + 256.5)
itmp    ftgen ift, 0, 4096, -30, 5, 1, inumh
inote   = inote + 1
if (inote < 127.5) igoto loop0

instr 1

kcps    expon 20, p3, 16000
kft     = int(256.5 + 69 + 12 * log(kcps / 440) / log(2))
kft     = (kft > 383 ? 383 : kft)

a1      phasor kcps
a1      tableikt a1, kft, 1, 0, 1

out a1 * 10000

endin
instr 2
```

```
kcps      expon 20, p3, 16000
kft       = int(256.5 + 69 + 12 * log(kcps / 440) / log(2))
kft       = (kft > 383 ? 383 : kft)

kgdur     limit 10 / kcps, 0.1, 1
a1        grain2 kcps, 0.02, kgdur, 30, kft, 6, -0.5

          out a1 * 2000

          endin

-----
partition :
-----

t 0 60
i 1 0 10
i 2 12 10
e
```

## Crédits

Auteur : Rasmus Ekman

Programmeur : Istvan Varga

Nouveau dans la version 4.17

# GEN33

GEN33 — Génère des formes d'onde complexes en mélangeant des sinus.

## Description

Ces routines génèrent des formes d'onde complexes en mélangeant des sinus, comme *GEN09*, mais les paramètres des partiels sont spécifiés dans une table déjà existante, ce qui permet de calculer n'importe quel nombre de partiels dans l'orchestre.

La différence entre *GEN33* et *GEN34* est que *GEN33* utilise la TFR inverse pour générer la sortie, alors que *GEN34* est basé sur l'algorithme utilisé dans les opcode oscils. *GEN33* ne permet que des partiels entiers, et ne supporte pas les tailles de table égales à une puissance-de-deux plus 1, mais peut être significativement plus rapide avec un grand nombre de partiels. D'un autre côté, avec *GEN34*, il est possible d'utiliser des numéros de partiel non entiers et un point de garde, et cette routine peut être plus rapide s'il n'y a qu'un petit nombre de partiels (noter que *GEN34* est aussi plusieurs fois plus rapide que *GEN09*, bien que ce dernier soit plus précis).

## Syntaxe

```
f # date taille 33 src nh ech [fmode]
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de deux et au moins 4.

*src* -- numéro de la table source. Cette table contient les paramètres de chaque partiel dans le format suivant :

*ampa*, *pna*, *phsa*, *ampb*, *pnb*, *phsb*, ...

les paramètres sont :

- *ampa*, *ampb*, etc. : amplitude relative des partiels. L'amplitude actuelle dépend de la valeur de *ech*, ou de la normalisation (si celle-ci est active).
- *pna*, *pnb*, etc. : numéro de partiel, ou fréquence, en fonction de *fmode* (voir ci-dessous) ; zéro et des valeurs négatives sont autorisés, cependant, si la valeur absolue du numéro de partiel dépasse (*taille* / 2), le partiel ne sera pas rendu. Avec *GEN33*, le numéro de partiel est arrondi à l'entier le plus proche.
- *phsa*, *phsb*, etc. : phase initiale, dans l'intervalle de 0 à 1.

La longueur de la table (sans compter le point de garde) devrait être d'au moins  $3 * nh$ . Si la table est trop courte, le nombre de partiels (*nh*) est réduit à (longueur de la table) / 3, arrondi vers zéro.

*nh* -- nombre de partiels. Zéro ou des valeurs négatives sont autorisés, et donnent une table vide (silence). Le nombre effectif peut être diminué si la table source (*src*) est trop courte, ou si certains partiels ont une fréquence trop haute.

*ech* -- échelle d'amplitude.

*fmode* (facultatif, défaut = 0) -- une valeur non nulle indique que les fréquences sont en Hz au lieu de numéros de partiel dans la table source. Le taux d'échantillonnage est supposé être *fmode* si celui-ci est

positif, ou  $-(sr * fmode)$  si une valeur négative est spécifiée.

## Exemples

```
; partiels 1, 4, 7, 10, 13, 16, etc. avec une fréquence de base de 400 Hz

ibsfrq = 400
; nombre de partiels estimé
inumh = int(1.5 + sr * 0.5 / (3 * ibsfrq))
; longueur de la table source
isrcln = int(0.5 + exp(log(2) * int(1.01 + log(inumh * 3) / log(2))))
; crée une table source vide
itmp   ftgen 1, 0, isrcln, -2, 0
ifpos  = 0
ifrq   = ibsfrq
inumh  = 0
11:
    tableiw ibsfrq / ifrq, ifpos, 1          ; amplitude
    tableiw ifrq, ifpos + 1, 1              ; fréquence
    tableiw 0, ifpos + 2, 1                 ; phase
ifpos  = ifpos + 3
ifrq   = ifrq + ibsfrq * 3
inumh  = inumh + 1
if (ifrq < (sr * 0.5)) igoto 11

; stocke la sortie dans la ftable 2 (taille = 262144)

itmp   ftgen 2, 0, 262144, -33, 1, inumh, 1, -1
```

## Voir Aussi

GEN09, GEN34

## Crédits

Programmeur : Istvan Varga  
Mars 2002

Nouveau dans la version 4.19

# GEN34

GEN34 — Génère des formes d'onde complexes en mélangeant des sinus.

## Description

Ces routines génèrent des formes d'onde complexes en mélangeant des sinus, comme *GEN09*, mais les paramètres des partiels sont spécifiés dans une table déjà existante, ce qui permet de calculer n'importe quel nombre de partiels dans l'orchestre.

La différence entre *GEN33* et *GEN34* est que *GEN33* utilise la TFR inverse pour générer la sortie, alors que *GEN34* est basé sur l'algorithme utilisé dans les opcode oscils. *GEN33* ne permet que des partiels entiers, et ne supporte pas les tailles de table égales à une puissance-de-deux plus 1, mais peut être significativement plus rapide avec un grand nombre de partiels. D'un autre côté, avec *GEN34*, il est possible d'utiliser des numéros de partiel non entiers et un point de garde, et cette routine peut être plus rapide s'il n'y a qu'un petit nombre de partiels (noter que *GEN34* est aussi plusieurs fois plus rapide que *GEN09*, bien que ce dernier soit plus précis).

## Syntaxe

```
f # date taille 34 src nh ech [fmode]
```

## Initialisation

*size* -- nombre de points dans la table. Doit être une puissance de deux ou une puissance-de-deux plus 1.

*src* -- numéro de la table source. Cette table contient les paramètres de chaque partiel dans le format suivant :

*ampa*, *pna*, *phsa*, *ampb*, *pnb*, *phsb*, ...

les paramètres sont :

- *ampa*, *ampb*, etc. : amplitude relative des partiels. L'amplitude actuelle dépend de la valeur de *ech*, ou de la normalisation (si celle-ci est active).
- *pna*, *pnb*, etc. : numéro de partiel, ou fréquence, en fonction de *fmode* (voir ci-dessous) ; zéro et des valeurs négatives sont autorisés, cependant, si la valeur absolue du numéro de partiel dépasse (*taille* / 2), le partiel ne sera pas rendu.
- *phsa*, *phsb*, etc. : phase initiale, dans l'intervalle de 0 à 1.

La longueur de la table (sans compter le point de garde) devrait être d'au moins  $3 * nh$ . Si la table est trop courte, le nombre de partiels (*nh*) est réduit à (longueur de la table) / 3, arrondi vers zéro.

*nh* -- nombre de partiels. Zéro ou des valeurs négatives sont autorisés, et donnent une table vide (silence). Le nombre effectif peut être diminué si la table source (*src*) est trop courte, ou si certains partiels ont une fréquence trop haute.

*ech* -- échelle d'amplitude.

*fmode* (facultatif, défaut = 0) -- une valeur non nulle indique que les fréquences sont en Hz au lieu de numéros de partiel dans la table source. Le taux d'échantillonnage est supposé être *fmode* si celui-ci est



positif, ou  $-(sr * fmode)$  si une valeur négative est spécifiée.

## Exemples

```
; partiels 1, 4, 7, 10, 13, 16, etc. avec une fréquence de base de 400 Hz
ibsfrq = 400
; nombre de partiels estimé
inumh = int(1.5 + sr * 0.5 / (3 * ibsfrq))
; longueur de la table source
isrcln = int(0.5 + exp(log(2) * int(1.01 + log(inumh * 3) / log(2))))
; crée une table source vide
itmp   ftgen 1, 0, isrcln, -2, 0
ifpos  = 0
ifrq   = ibsfrq
inumh  = 0
11:
    tableiw ibsfrq / ifrq, ifpos, 1          ; amplitude
    tableiw ifrq, ifpos + 1, 1              ; fréquence
    tableiw 0, ifpos + 2, 1                 ; phase
ifpos  = ifpos + 3
ifrq   = ifrq + ibsfrq * 3
inumh  = inumh + 1
if (ifrq < (sr * 0.5)) igoto 11

; stocke la sortie dans la ftable 2 (taille = 262144)
itmp   ftgen 2, 0, 262144, -34, 1, inumh, 1, -1
```

## Voir Aussi

GEN09, GEN33

## Crédits

Programmeur : Istvan Varga  
Mars 2002

Nouveau dans la version 4.19

# GEN40

GEN40 — Génère une distribution aléatoire à partir d'un histogramme.

## Description

Génère une distribution aléatoire continue en partant de la forme d'un histogramme défini par l'utilisateur.

## Syntaxe

```
f # date taille 40 tblforme
```

## Exécution

La forme de l'histogramme doit être stockée dans une table préalablement définie, en fait, *tblforme* doit contenir le numéro de cette table.

La forme de l'histogramme peut être générée avec n'importe quelle GEN routine. Comme il n'y a pas d'interpolation lorsque GEN40 opère la traduction, il est suggéré de donner à la table contenant la forme de l'histogramme une taille raisonnablement grande, afin d'obtenir une meilleure précision (cependant, cette dernière table peut être détruite après le traitement pour récupérer de la mémoire).

Ce sous-programme est prévu pour être utilisé avec l'opcode *cusernd* (voir *cusernd* pour plus d'information).

## Crédits

Auteur : Gabriel Maldonado

# GEN41

GEN41 — Génère une liste aléatoire de paires numériques.

## Description

Génère une fonction de distribution aléatoire discrète en donnant une liste de paires numériques.

## Syntaxe

```
f # date taille -41 valeur1 prob1 valeur2 prob2 valeur3 prob3 ... valeurN probN
```

## Exécution

Le premier nombre de chaque paire est une valeur, et le second est la probabilité que cette valeur soit choisie par un algorithme aléatoire. Même si n'importe quel nombre peut être assigné à l'élément probabilité de chaque paire, il vaut mieux lui donner une valeur en pourcentage, afin de rendre les choses plus claires pour l'utilisateur.

Ce sous-programme est prévu pour être utilisé avec les opcodes *dusernd* et *urd* (voir *dusernd* pour plus d'information).

## Crédits

Auteur : Gabriel Maldonado

# GEN42

GEN42 — Génère une distribution aléatoire d'intervalles discrets de valeurs.

## Description

Génère une fonction de distribution aléatoire d'intervalles discrets de valeurs en donnant une liste de groupes de trois nombres.

## Syntaxe

```
f # date taille -42 min1 max1 prob1 min2 max2 prob2 min3 max3 prob3 ... minN maxN probN
```

## Exécution

Le premier nombre de chaque groupe est la valeur minimum de l'intervalle, le second est la valeur maximum et le troisième est la probabilité qu'un élément appartenant à cet intervalle de valeurs soit choisi par un algorithme aléatoire. La probabilité pour un intervalle doit être une fraction de 1, et la somme des probabilités pour tous les intervalles doit être égale à 1.

Ce sous-programme est prévu pour être utilisé avec les opcodes *duserrnd* et *urd* (voir *duserrnd* pour plus d'information). Comme ni *duserrnd* ni *urd* n'utilisent l'interpolation, il est suggéré de donner une taille raisonnablement grande.

## Crédits

Auteur : Gabriel Maldonado

# GEN43

GEN43 — Charge un fichier PVOCEX contenant une analyse VP.

## Description

Ce sous-programme charge un fichier PVOCEX contenant l'analyse VP (amp-fréq) d'un fichier son et calcule les magnitudes moyennes de toutes les trames d'analyse d'un ou de tous les canaux audio. Il crée ensuite une table avec ces magnitudes pour chaque bin VP.

## Syntaxe

```
f # date taille 43 codfic canal
```

## Initialisation

*taille* -- nombre de points dans la table, puissance de deux ou puissance-de-deux plus 1. *GEN43* ne fait aucune distinction entre ces deux tailles, mais la table doit avoir pour taille au moins la moitié de celle de la tfr. Les bins VP couvrent le spectre positif de 0 Hz (index 0 de la table) à la fréquence de Nyquist (index  $tailletfr/2+1$  de la table) par incréments réguliers (de taille  $sr/tailletfr$ ).

*codfic* -- un fichier pvocex (qui peut être généré par pvanal).

*canal* -- numéro du canal audio duquel les magnitudes seront extraites ; un 0 donnera la moyenne des magnitudes de tous les canaux.

La lecture s'arrête à la fin du fichier.



### Note

Si p4 est positif, la table sera post-normalisée. Un p4 négatif empêchera la post-normalisation.

## Exemples

```
f1 0 512 43 "viola.pvx" 1
f1 0 -1024 -43 "noiseprint.pvx" 0
```

On peut utiliser cette table comme table de masquage pour *pvstencil* et *pvsmaska*. Le premier exemple utilise un fichier d'analyse de vocodeur de phase par TFR à 1024 points duquel on utilise le premier canal. Le second utilise tous les canaux d'un fichier de 2048 points, sans post-normalisation. Pour les applications à la réduction de bruit avec *pvstencil*, il est mieux de ne pas normaliser la table (code GEN négatif).

## Crédits

Auteur : Victor Lazzarini

# GEN49

GEN49 — Transfère les données d'un fichier son MP3 dans une table de fonction.

## Description

Ce sous-programme transfère les données d'un fichier son MP3 dans une table de fonction.

## Syntaxe

```
f# time size 49 filcod skiptime format
```

## Exécution

*size* -- nombre de points dans la table. Ordinairement une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*) ; la taille de table maximale est de 16777216 ( $2^{24}$ ) points. L'allocation de mémoire pour la table peut être *différée* en mettant ce paramètre à 0 ; la taille allouée est alors le nombre de points dans le fichier (probablement pas une puissance de 2), et la table n'est pas utilisable par les oscillateurs normaux, mais par l'unité *loscil*. Le fichier son peut être mono ou stéréo.

*filcod* -- entier ou chaîne de caractères dénotant le nom du fichier son source. Un entier dénote le fichier *soundin.filcod* ; une chaîne de caractères (entre apostrophes doubles, espaces autorisés) donne le nom du fichier lui-même, optionnellement un nom de chemin complet. Si le chemin n'est pas complet, le fichier est d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) enfin par SFDIR. Voir aussi *soundin*.

*skiptime* -- commence à lire à *skiptime* secondes dans le fichier.

*format* -- spécifie le format de fichier audio requis :

1 - Fichier mono	3 - Premier canal (gauche)
2 - Fichier stéréo	4 - Second canal (droite)

Si *format* = 0 le format d'échantillon est pris dans l'en-tête du fichier son.



### Note

- La lecture s'arrête à la fin du fichier ou lorsque la table est pleine. Les cellules de la table non remplies contiendront des zéros.
- Si p4 est positif, la table sera post-normalisée (reproportionnée avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.

## Exemples

Voici un exemple simple de la routine GEN49. Il utilise les fichiers *gen49.csd* [examples/gen49.csd] et

*beats.mp3* [examples/beats.mp3]. Il utilise le fichier MP3 « beats.mp3 ».

### Exemple 650. Un exemple simple de la routine GEN49.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o gen01.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  kamp = 30000
  kcps = 1
  ifn = 1

  ; Play the audio sample stored in Table #1.
  al oscil kamp, kcps, ifn
  out al
endin

</CsInstruments>
<CsScore>

; Table #1: read an audio file (using GEN49).
f 1 0 131072 49 "beats.mp3" 0 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Ecrit par John ffitc

Février 2009.

# GEN51

GEN51 — Ce sous-programme remplit une table avec une échelle microtonale personnalisée, à la manière des opcodes de Csound *cpstun*, *cpstuni* et *cpstmid*.

## Description

Ce sous-programme remplit une table avec une échelle microtonale personnalisée, à la manière des opcodes de Csound *cpstun*, *cpstuni* et *cpstmid*.

## Syntaxe

```
f # date taille -51 nbrdegrés intervalle fréqbase touchebase rapport1 rapport2 .... rapportN
```

## Exécution

Les quatre premiers paramètres (c'est-à-dire p5, p6, p7 et p8) définissent les directives de génération suivantes :

*p5 (nbrdegrés)* -- le nombre de degrés de l'échelle microtonale

*p6 (intervalle)* -- l'intervalle de fréquences couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.

*p7 (fréqbase)* -- la fréquence de base de l'échelle en cps

*p8 (touchebase)* -- L'indice entier dans la table auquel assigner la fréquence de base inchangée

Les autres paramètres définissent les rapports de l'échelle :

*p9 ... pN (rapport1 ... etc.)* -- les rapports des degrés de l'échelle

Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 cps assignée à la touche numéro 60, l'instruction f de la partition pour générer la table serait :

```
;          nbrdegrés      fréqbase      rapports (tempérament égal) .....  
;          intervalle      touchebase  
f1 0 64 -51      12      2      261      60      1 1.059463 1.12246 1.18920 ..etc...
```

Après le calcul du gen, la table f1 est remplie avec 64 valeurs de fréquences différentes. Le 60ème élément est rempli avec la valeur de fréquence 261, et tous les autres éléments de la table (précédents et suivants) sont remplis selon les rapports des degrés.

Un autre exemple avec une échelle de 24 degrés, une fréquence de base de 440 cps assignée à la touche numéro 48, et un intervalle de répétition de 1,5 :

```
;          nbrdegrés      fréqbase      rapports .....  
;          intervalle      touchebase  
f1 0 64 -51      24      1.5      440      48      1 1.01 1.02 1.03 ..etc...
```

## Crédits



Auteur : Gabriel Maldonado

# GEN52

GEN52 — Crée une table à plusieurs canaux entrelacés à partir des tables source spécifiées, dans le format attendu par l'opcode *ftconv*.

## Description

GEN52 crée une table à plusieurs canaux entrelacés à partir des tables source spécifiées, dans le format attendu par l'opcode *ftconv*. Il peut aussi être utilisé pour extraire un canal d'une table multicanaux et le stocker dans une table mono normale, copier des tables en omettant certains échantillons, ajouter un décalai, ou stocker en ordre inverse, etc.

Il faut donner trois paramètres pour chaque canal à traiter. *fsrc* déclare le numéro de la f-table source. Le paramètre *offset* spécifie un décalage pour le fichier source. S'il est différent de 0, le fichier source n'est pas lu depuis le début, un nombre *offset* de valeurs étant ignorées. L'*offset* est utilisé pour déterminer le numéro de canal à lire depuis les f-tables entrelacées, par exemple pour le canal 2, *offset* doit valoir 1. Il peut aussi être utilisé pour fixer un décalage de lecture sur la table source. Ce paramètre donne des valeurs absolues, si bien que si l'on désire un décalage de 20 unités d'échantillonnage pour une f-table à deux canaux, *offset* doit valoir 40. Le paramètre *srcchnls* est utilisé pour fixer le nombre de canaux dans la f-table source. Ce paramètre fixe la taille du pas de progression lors de la lecture de la f-table source.

Quand il y a plus d'un canal (*nchannels* > 1), les f-tables source sont entrelacées dans la table nouvellement créée.

Si la f-table source est finie avant que la f-table destination ne soit remplie, les valeurs restantes sont fixées à 0.

## Syntaxe

```
f # date taille 52 ncanaux fsrcl offset1 srcchnls1 [fsrc2 offset2 srcchnls2 ... fsrcN offsetN srcchnlsN]
```

## Exemple

```
; tables sources
f 1 0 16384 10 1
f 2 0 16384 10 0 1
; crée une table avec 2 canaux entrelacés
f 3 0 32768 -52 2 1 0 1 2 0 1
; extrait le premier canal de la table 3
f 4 0 16384 -52 1 3 0 2
; extrait le second canal de la table 3
f 5 0 16384 -52 1 3 1 2
```

## Crédits

Auteur : Istvan Varga

---

# Les Programmes Utilitaires

Dan Ellis, MIT Media Lab

Les utilitaires de Csound sont des programmes de *prétraitement de fichier son* qui retournent de l'information sur un fichier son ou qui créent une version d'analyse de celui-ci à utiliser par certains générateurs de Csound. Bien que destinés à différents usages, ils ont en commun le mécanisme d'accès au fichier son et sont descriptibles comme un ensemble. Les programmes Utilitaires de Fichiers Son peuvent être appelés de deux manières équivalentes :

```
csound [-U nomutilitaire] [options] [noms_fichier]
```

```
nomutilitaire [options] [noms_fichier]
```

Dans le premier cas, l'utilitaire est appelé comme une partie de l'exécutable de Csound, tandis que dans le second il est appelé comme un programme autonome. Le second est plus petit d'environ 200 K, mais les deux formes fonctionnent de manière identique. La première est pratique pour éviter la maintenance et l'utilisation de plusieurs programmes indépendants - un programme fait tout. Quand on utilise cette forme, un *drapeau -U* détecté dans la ligne de commande provoquera l'interprétation des options et des noms suivants comme ceux de l'utilitaire nommé ; cela signifie que le mécanisme de génération de Csound ne sera pas invoqué et que le programme se terminera à la fin du traitement par l'utilitaire.

## Répertoires.

Les noms de fichier sont de deux sortes, fichiers son sources et fichiers d'analyse résultants. Chacun a une convention de nommage hiérarchique, influencée par le répertoire depuis lequel l'utilitaire est appelé. Les fichiers son sources avec un nom de chemin complet (commençant par un point (.), une barre oblique (/), ou pour ThinkC incluant un deux-points (:)), ne seront cherchés que dans le répertoire nommé. Les fichiers son sans chemin seront recherchés d'abord dans le répertoire courant, ensuite dans le répertoire nommé par la variable d'environnement SSDIR (si elle est définie), ensuite dans le répertoire nommé par SFDIR. Une recherche infructueuse retournera une erreur "cannot open".

Les fichiers d'analyse résultants sont écrits dans le répertoire courant, ou le répertoire nommé si un chemin est inclus. Pour être ordonné, il est bon de séparer les fichiers d'analyse des fichiers son, habituellement dans un répertoire différent référencé par la variable d'environnement SADIR. Il est commode de lancer l'analyse depuis le répertoire SADIR. Quand un fichier d'analyse est invoqué ultérieurement par un générateur de Csound il est cherché en premier dans le répertoire courant, puis dans le répertoire défini par SADIR.

## Formats des Fichiers Son.

Csound peut lire et écrire des fichiers audio dans différents formats. Les formats d'écriture sont décrits par des options de la commande Csound. En lecture, le format est déterminé par l'en-tête du fichier, et les données sont automatiquement converties en virgule flottante pendant le traitement interne. Quand Csound est installé sur un hôte qui a des conventions de fichier son locales (SUN, NeXT, Macintosh) il peut comprendre de manière conditionnelle du code local qui crée des fichiers son non portables vers d'autres hôtes. Cependant, sur tous les hôtes, Csound peut toujours générer et lire des fichiers du type AIFF, qui est ainsi un format portable. Les bibliothèques de sons échantillonnés sont typiquement en AIFF, et la variable d'environnement SSDIR pointe habituellement vers un répertoire contenant de tels sons. S'il est défini, le répertoire SSDIR fait partie des chemins de recherche pour l'accès aux fichiers son. Noter que certains sons échantillonnés AIFF ont un mécanisme de boucle audio pour

les notes tenues ; les programmes d'analyse ne parcourent les segments de boucle qu'une fois.

Pour les fichiers son sans en-tête, une valeur *SR* peut être fournie par l'*option -r* (ou sa valeur par défaut). Si l'*en-tête SR* et l'option de ligne de commande sont tous deux présents, la valeur de l'option remplacera l'en-tête.

Quand les programmes d'Analyse accèdent à un son, un seul canal est lu. Pour les fichiers stéréo ou quadro, le canal par défaut est le canal un ; d'autres canaux peuvent être obtenus à la demande.

## Génération d'un Fichier d'Analyse (ATSA, CVANAL, HETRO, LPANAL, PVANAL)

Les utilitaires suivants existent pour l'analyse d'un Fichier Son :

- *ATSA* : Analyse ATS à utiliser avec les opcodes de Csound de *Resynthèse ATS*.
- *CVANAL* : Analyse de Fourier d'une Réponse Impulsionnelle pour l'opérateur *convolve*.
- *HETRO* : Analyse hétérodyne pour le générateur de Csound *adsyn*.
- *LPANAL* : Analyse de codage prédictif linéaire pour les opcodes de Csound de *Resynthèse par Codage Prédictif Linéaire (LPC)*.
- *PVANAL* : Analyse par vocodeur de phase pour le générateur de Csound *pvoc*.

## atsa

atsa — Effectue une analyse ATS sur un fichier son.

## Description

Analyse ATS à utiliser avec les opcodes de Csound de *Resynthèse ATS*.

## Syntaxe

```
csound -U atsa [options] nomfichier_entree nomfichier_sortie
```

## Initialisation

Les options suivantes peuvent être positionnées pour atsa. (Les valeurs par défaut sont mises entre parenthèses) :

- b début (0,000000 secondes)
- e durée (0,000000 secondes, signifie jusqu'à la fin)
- l fréquence la plus basse (20,000000 Hz)
- H fréquence la plus haute (20000,000000 Hz)
- d déviation en fréquence (0,100000 de la fréquence d'un partiel)
- c cycles par fenêtre (4 cycles)
- w type de fenêtre (type : 1) (Options : 0=BLACKMAN, 1=BLACKMAN\_H, 2=HAMMING, 3=VONHANN)
- h taille de saut (0,250000 de la taille de fenêtre)
- m magnitude la plus faible (-60,000000)
- t longueur de trajectoire (3 trames)
- s longueur minimale de segment (3 trames)
- g longueur minimale des blancs (3 trames)
- T seuil du SMR (30,000000 dB SPL)
- S SMR Minimum de Segment (60,000000 dB SPL)
- P contribution du dernier pic (0,000000 des paramètres du dernier pic)
- M contribution du SMR (0,500000)
- F Type de Fichier (type : 4) (Options : 1=amp. et fréq. seulement, 2=amp., fréq. et phase, 3=amp., fréq. et résiduel, 4=amp., fréq., phase et résiduel)

## Paramètres

L'analyse ATS a été conçue par Juan Pampin. Pour une information complète sur ATS visiter : <http://www-ccrma.stanford.edu/~juan/ATS.html>.

Les paramètres d'analyse doivent être réglés soigneusement pour l'Algorithme d'Analyse (ATSA) afin de capturer la nature du signal à analyser. Comme ils sont nombreux, ATSH offre la possibilité de les Sauvegarder/Charger dans un Fichier Binaire portant l'extension ".apf". L'extension n'est pas obligatoire, mais recommandée. Une brève explication de chaque Paramètre d'Analyse suit :

1. Début (secs.): la date de début de l'analyse en secondes.
2. Durée (secs.): la durée de l'analyse en secondes. Un zéro signifie la durée entière du fichier son en entrée.
3. Fréquence la Plus Basse (Hz) : ce paramètre va déterminer partiellement la taille de la Fenêtre d'Analyse à utiliser. Pour calculer la taille de la Fenêtre d'Analyse, la période de la Fréquence la Plus Basse en échantillons ( $SR / LF$ ) est multipliée par le nombre de cycles de celle-ci que l'utilisateur veut

caser dans la Fenêtre d'Analyse (voir paramètre 6). Cette valeur est arrondie à la plus proche puissance de deux supérieure pour déterminer la taille de la TFR pour l'analyse. Les échantillons en trop sont remplis par des zéros. Si le signal est un son unique, harmonique, alors la valeur de la Fréquence la Plus Basse sera celle du fondamental ou d'un sous-harmonique de celui-ci. Si le son n'est pas harmonique, alors sa fréquence significative la plus basse pourra être une bonne valeur de départ.

4. Fréquence la Plus Haute (Hz) : fréquence la plus haute à prendre en compte pour la Détection de Pic. Une fois que l'on sait qu'aucune information pertinente ne se trouve au-delà d'une certaine fréquence, l'analyse peut être plus rapide et plus précise en réglant la Fréquence la Plus Haute à cette valeur.
5. Déviation de Fréquence (Rapport) : déviation de fréquence autorisée pour chaque pic dans l'Algorithme de Continuation des Pics, comme fraction de la fréquence concernée. Par exemple, si l'on considère un pic à 440 Hz et une Déviation de 0,1 l'Algorithme de Continuation des Pics n'essayera de trouver des candidats pour la continuité qu'entre 396 et 484 Hz (10% au-dessus et en-dessous de la fréquence du pic). Une petite valeur produira probablement plus de trajectoires tandis qu'une grande valeur les réduira, mais au prix d'une plus grande difficulté à traiter l'information par la suite.
6. Nombre de Cycles de la Fréquence la Plus Basse à caser dans une Fenêtre d'Analyse : il déterminera aussi partiellement la taille de la Fenêtre de Transformation de Fourier à utiliser. Voir le paramètre 3. Pour des signaux à un seul harmonique, il est supposé être supérieur à 1 (typiquement 4).
7. Taille de Saut (Rapport) : taille de l'intervalle entre une Fenêtre d'Analyse et la suivante exprimée comme une fraction de la Taille de Fenêtre. Par exemple, une Taille de Saut de 0,25 "sautera" de 512 échantillons (les Fenêtres se chevaucheront sur 75% de leur taille). Ce paramètre déterminera aussi la taille des trames d'analyse obtenues. Les signaux qui changent leur spectre très rapidement (comme les sons de la Parole) peuvent nécessiter un taux de trame élevé afin de suivre au mieux leurs changements.
8. Limite d'Amplitude (dB) : la valeur d'amplitude la plus élevée à prendre en compte pour la Détection de Pic.
9. Type de Fenêtre : la forme de la fonction de lissage à utiliser pour l'Analyse de Fourier. Il y a quatre choix possibles pour le moment : Blackman, Blackman-Harris, Von Hann, et Hanning. Des spécifications précises sur celles-ci se trouvent facilement dans la bibliographie sur le traitement numérique du signal.
- 10 Longueur de Trajectoire (Trames) : L'Algorithme de Continuation des Pics regardera "en arrière" sur un nombre de trames égal à Longueur afin de réaliser sa tâche au mieux, et d'éviter que les trajectoires de fréquence ne s'incurvent trop et perdent leur stabilité. Cependant, une grande valeur pour ce paramètre ralentira l'analyse de manière significative.
- 11 Longueur Minimale de Segment (Trames) : une fois l'analyse réalisée, les données spectrales peuvent être "nettoyées" durant le post-traitement. Les trajectoires plus petites que cette valeur sont supprimées si leur SMR moyen est inférieur au SMR Minimum de Segment (voir les paramètres 16 et 14). Ceci peut aider à éviter les changements soudains non pertinents tout en gardant un taux de trames élevé, réduisant aussi le nombre de sinusoides épisodiques durant la synthèse.
- 12 Longueur Minimale des Blancs (Trames): comme le paramètre 11, celui-ci est aussi utilisé pour nettoyer les données durant le post-traitement. Dans ce cas, les blancs (valeurs d'amplitude nulle, c'est-à-dire le "silence" théorique) contigus dont le nombre de trames est plus grand que Longueur sont remplis avec des valeurs d'amplitude/fréquence obtenues par interpolation linéaire des trames actives adjacentes. Ce paramètre empêche les interruptions soudaines des trajectoires stables tout en gardant un taux de trames élevé.
- 13 Seuil du SMR (dB SPL) : également un paramètre de post-traitement, le seuil du SMR est utilisé pour éliminer les partiels avec de faibles moyennes.

14 SMR Minimum de Segment (dB SPL) : ce paramètre est utilisé en combinaison avec le paramètre 11.

- Les segments courts ayant un SMR moyen inférieur à cette valeur seront supprimés durant le post-traitement.

15 Contribution du Dernier Pic (0 à 1) : comme c'est expliqué dans le Paramètre 10, l'Algorithme de Continuation des Pics regarde "en arrière" sur plusieurs trames afin de réaliser sa tâche au mieux. Ce paramètre aidera à pondérer la contribution du premier des pics précédents sur les autres. Une valeur de zéro signifie que tous les pics précédents (jusqu'à la taille du Paramètre 10) sont pris également en compte.

16 Contribution du SMR (0 à 1) : en plus de la proximité en fréquence des pics, l'Algorithme de Continuation des Pics ATS peut utiliser une information psychoacoustique (le Rapport Signal-Masque, ou SMR) pour améliorer les résultats perceptifs. Ce paramètre indique quelle quantité d'information SMR est utilisée durant la détection. Par exemple, une valeur de 0,5 fait que l'Algorithme de Continuation des Pics utilise 50% d'information SMR et 50% d'information de Proximité en Fréquence pour décider quel est le meilleur candidat pour continuer la trajectoire sinusoïdale.

## Exemples

La commande suivante :

```
atsa -b0.1 -e1 -l100 -H10000 -w2 fichieraudio.wav fichieraudio.ats
```

Génère le fichier d'analyse ATS 'fichieraudio.ats' à partir du fichier original 'fichieraudio.wav'. L'analyse commence à partir de 0,1 seconde dans le fichier et elle est effectuée sur 1 seconde. La fréquence la plus basse est 100 Hz et la plus haute est 10 kHz. Une fenêtre de Hamming est utilisée pour chaque trame d'analyse.

## cvanal

**cvanal** — Convertit un fichier son en une trame de transformée de Fourier.

### Description

Analyse de Fourier d'une Réponse Impulsionnelle pour l'opérateur *convolve*

### Syntaxe

```
csound -U cvanal [options] nomfichier_entree nomfichier_sortie
```

```
cvanal [options] nomfichier_entree nomfichier_sortie
```

### Initialisation

*cvanal* -- convertit un fichier son en une trame de transformée de Fourier. Le fichier de sortie peut être utilisé par l'opérateur *convolve* pour réaliser une Convolution Rapide entre un signal d'entrée et la réponse impulsionnelle originale. L'analyse est conditionnée par les options ci-dessous. Un espace est facultatif entre le drapeau et son argument.

*-s srate* -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur *srate* de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

*-c canal* -- numéro du canal à traiter. S'il est omis, tous les canaux sont traités par défaut. Si une valeur est donnée, seul le canal choisi sera traité.

*-b début* -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

*-d durée* -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier.

### Exemples

```
cvanal unson fichiercv
```

analysera le fichier son "unson" pour produire le fichier "fichiercv" à utiliser avec *convolve*.

Pour utiliser des données qui ne sont pas déjà contenues dans un fichier son, un convertisseur de fichier son qui accepte des fichiers texte peut être utilisé pour créer un fichier audio standard, par exemple le format .DAT pour SOX. Ceci est utile pour implémenter des filtres RIF.

### Fichiers

Le fichier de sortie a un en-tête spécial *convolve*, contenant les détails du fichier source audio. Les données d'analyse sont stockées comme des nombres « virgule flottante », en forme rectangulaire (réel/imaginaire).



#### Note

Le fichier d'analyse n'est *pas* indépendant du système ! Assurez-vous que les données originales de la réponse impulsionnelle sont conservées. Si nécessaire, le fichier d'analyse pourra être recréé.



## Crédits

Auteur : Greg Sullivan

Basé sur l'algorithme donné dans *Elements Of Computer Music*, par F. Richard Moore.

## hetro

hetro — Décompose un fichier son en entrée en composantes sinusoïdales.

### Description

Analyse par filtre hétérodyne pour le générateur de Csound *adsyn*.

### Syntaxe

```
csound -U hetro [options] nomfichier_entree nomfichier_sortie
```

```
hetro [options] nomfichier_entree nomfichier_sortie
```

### Initialisation

*hetro* prend un fichier son en entrée, le décompose en composantes sinusoïdales, et sort une description de ces composantes sous la forme de pistes de points charnière d'amplitude et de fréquence. L'analyse est conditionnée par les options de contrôle ci-dessous. Un espace est facultatif entre drapeau et argument.

*-s srate* -- taux d'échantillonnage du fichier audio en entrée. Il remplacera la valeur *srate* de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000. Noter que pour la synthèse *adsyn* le taux d'échantillonnage du fichier source et de l'orchestre générateur n'ont pas à être les-mêmes.

*-c canal* -- numéro du canal à traiter. La valeur par défaut est 1.

*-b début* -- date de début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

*-d durée* -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier. La longueur maximale est de 32,766 secondes.

*-f freqdeb* -- fréquence de départ estimée du fondamental, nécessaire pour initialiser l'analyse par le filtre. La valeur par défaut est 100 (cps).

*-h partiels* -- nombre d'harmoniques recherchés dans le fichier audio. La valeur par défaut est 10, la valeur maximale dépend de la mémoire disponible.

*-M ampmax* -- amplitude maximale obtenue par addition sur toutes les pistes simultanées. La valeur par défaut est 32767.

*-m ampmmin* -- seuil d'amplitude en-dessous duquel une paire de pistes amplitude/fréquence sera considérée comme inactive et ne contribuera pas à la somme en sortie. Valeurs typiques : 128 (48 dB en-dessous de l'échelle complète, 64 (54 dB en-dessous), 32 (60 dB en-dessous), 0 (pas de seuillage). Le seuil par défaut est 64 (54 dB en-dessous).

*-n brkpts* -- nombre initial de points charnière de l'analyse dans chaque piste d'amplitude et de fréquence, avant le seuillage (*-m*) et la consolidation linéaire des points charnière. Les points initiaux sont répartis uniformément sur toute la durée. La valeur par défaut est 256.

*-l cutfreq* -- substitue un filtre passe-bas de Butterworth du 3ème ordre avec une fréquence de coupure *cutfreq* (en Hz), à la place du filtre par défaut qui est un filtre de moyenne en peigne. La valeur par défaut est 0 (ne pas utiliser).

### Exécution

A partir de Csound 4.08, *hetro* peut écrire des fichiers de sortie SDIF si le nom du fichier de sortie se termine par ".sdif" ou ".SDIF". Voir l'utilitaire *sdif2ad* pour plus d'information sur le support de SDIF dans Csound.

## Exemples

```
hetro -s44100 -b.5 -d2.5 -hl6 -M24000 fichieraudio.test adsynfile7
```

Ceci analyse 2,5 secondes du canal 1 du fichier "fichieraudio.test", enregistré à 44,1 kHz, commençant 0,5 secondes après le début, et place le résultat dans le fichier "adsynfile7". Nous ne voulons que les 16 premiers harmoniques du son, avec 256 points charnière par piste d'amplitude ou de fréquence, et un pic de la somme des amplitudes de 24000. Le fondamental est estimé au commencement à 100 Hz. Le seuil d'amplitude est de 54 dB en-dessous de l'échelle complète.

Le filtre passe-bas de Butterworth n'est pas activé.

## Format de Fichier

Le fichier de sortie contient des suites temporelles de valeurs d'amplitude et de fréquence pour chaque harmonique d'une source audio additive complexe. L'information se présente sous la forme de points charnière (date, valeur, date, valeur, ...) en utilisant des entiers sur 16 bit dans l'intervalle 0 - 32767. Le temps est donné en millisecondes, et les fréquences en Hz (cps). Les données des points charnières sont exclusivement non-négatives, et les valeurs -1 et -2 signifient uniquement le début de nouvelles pistes d'amplitude et de fréquence. Une piste se termine par la valeur 32767. Avant d'être écrite en sortie, chaque piste subit une réduction de données par seuillage d'amplitude et consolidation linéaire des points charnière.

Un composant harmonique est défini par deux ensembles de points charnière : un ensemble d'amplitudes, et un ensemble de fréquences. Dans un fichier composé ces ensembles peuvent apparaître dans n'importe quel ordre (amplitude, fréquence, amplitude ....; ou amplitude, amplitude, ..., puis fréquence, fréquence, ...). Durant la resynthèse par *adsyn* les ensembles sont automatiquement appariés (amplitude, fréquence) dans l'ordre dans lequel ils sont trouvés. Il doit y avoir un nombre égal de chaque sorte.

Un fichier de contrôle *adsyn* légal pourrait avoir le format suivant :

```
-1 temps1 valeur1 ... tempsK valeurK 32767 ; points charnière d'amplitude pour le partiel 1
-2 temps1 valeur1 ... tempsL valeurL 32767 ; points charnière de fréquence pour le partiel 1
-1 temps1 valeur1 ... tempsM valeurM 32767 ; points charnière d'amplitude pour le partiel 2
-2 temps1 valeur1 ... tempsN valeurN 32767 ; points charnière de fréquence pour le partiel 2
-2 temps1 valeur1 .....
-2 temps1 valeur1 ..... ; pistes appariables pour les partiels 3 et 4
-1 temps1 valeur1 .....
-1 temps1 valeur1 .....
```

## Crédits

Auteur : Tom Sullivan

1992

Auteur : John ffitich

1994

Auteur : Richard Dobson

2000

Octobre 2002. Merci à Rasmus Ekman, pour l'addition d'une note au sujet du format SDIF.

## lpanal

**lpanal** — Effectue une analyse par prédiction linéaire et par détection de hauteur sur un fichier son.

### Description

Analyse par prédiction linéaire pour les opcodes de Csound *Resynthèse par Codage Prédicatif Linéaire (LPC)*.

### Syntaxe

```
csound -U lpanal [options] nomfichier_entree nomfichier_sortie
```

```
lpanal [options] nomfichier_entree nomfichier_sortie
```

### Initialisation

*lpanal* effectue à la fois une analyse par lpc et par détection de hauteur sur un fichier son pour produire une suite ordonnée de *trames* d'information de contrôle appropriée pour la resynthèse avec Csound. L'analyse est conditionnée par les options de contrôle ci-dessous. Un espace est facultatif entre le drapeau et sa valeur.

**-a** -- [stockage alternatif] demande à *lpanal* d'écrire un fichier avec les valeurs des pôles du filtre plutôt que les fichiers de coefficients de filtre habituels. Quand *lpread* / *lpreson* sont utilisés avec des fichiers de pôles, une stabilisation automatique est effectuée et le filtre ne deviendra pas incontrôlable. (C'est le réglage par défaut dans la GUI Windows) - Changé par Marc Resibois.

**-s srate** -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur *srate* de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

**-c canal** -- numéro du canal à traiter. La valeur par défaut est 1.

**-b début** -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

**-d durée** -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier.

**-p npoles** -- nombres de pôles pour l'analyse. La valeur par défaut est 34, le maximum 50.

**-h taillesaut** -- taille du saut (en échantillons) entre les trames d'analyse. Détermine le nombre de trames par seconde (*srate* / *taillesaut*) dans le fichier de contrôle en sortie. La taille des trames d'analyse est de *taillesaut* \* 2 échantillons. La valeur par défaut est 200, le maximum 500.

**-C chaîne** -- texte pour le champ commentaire de l'en-tête du fichier lp. La valeur par défaut est une chaîne nulle.

**-P mincps** -- fréquence la plus basse (en Hz) pour la détection de hauteur. -P0 signifie pas de détection de hauteur.

**-Q maxcps** -- fréquence la plus haute (en Hz) pour la détection de hauteur. Plus l'intervalle de hauteurs est étroit, plus l'estimation de hauteur est précise. Les valeurs par défaut sont -P70, -Q200.

**-v verbosité** -- niveau d'information affiché sur le terminal pendant l'analyse.

- 0 = aucune

- 1 = verbeux
- 2 = débogage

La valeur par défaut est 0.

## Exemples

```
lpanal -a -p26 -d2.5 -P100 -Q400 fichieraudio.test lpfil22
```

analysera les premières 2,5 secondes du fichier "fichieraudio.test", produisant *srates* / 200 trames par seconde, chacune contenant les coefficients d'un filtre à 26 pôles et une estimation de hauteur entre 100 et 400 Hz. La sortie stabilisée (*-a*) sera placée dans "lpfil22" dans le répertoire courant.

## Format de Fichier

La sortie est un fichier constitué d'un en-tête identifiable plus un ensemble de trames de données d'analyse en virgule flottante. Chaque trame contient quatre valeurs d'information de hauteur et de gain, suivies par *npoles* coefficients de filtre. Le fichier est lisible par l'opcode *lpread* de Csound.

*lpanal* est une modification importante des programmes d'analyse lpc de Paul Lanksy.

## pvanal

pvanal — Convertit un fichier son en une série de trames de transformation de Fourier à court terme.

### Description

Analyse de Fourier pour le générateur de Csound *pvoc*

### Syntaxe

```
csound -U pvanal [options] nomfic_entree nomfic_sortie
```

```
pvanal [options] nomfic_entree nomfic_sortie
```

### Extension de pvanal pour créer un fichier PVOC-EX.

L'utilitaire standard de Csound pvanal a été étendu pour permettre la création d'un fichier au format PVOC-EX, en utilisant l'interface existante. Pour créer un fichier PVOC-EX, le nom de fichier doit avoir comme extension « .pvx », par exemple « test.pvx ». La nécessité pour la taille de TFR d'être une puissance de deux n'est plus obligatoire ici, et n'importe quelle valeur positive est acceptée ; les nombres impairs sont arrondis en interne. Cependant, les tailles en puissance de deux sont toujours préférables pour toutes les applications normales.

Les drapeaux de sélection de canal sont ignorés, et tous les canaux de la source seront analysés et écrits dans le fichier de sortie, jusqu'à la limite, fixée à la compilation, de huit canaux. La taille de la fenêtre d'analyse (itailfen) est fixée en interne au double de la taille de la TFR.

### Initialisation

*pvanal* convertit un fichier son en une série de trames de transformation de Fourier à court terme (STFT) à espacement temporel régulier (une représentation du domaine fréquentiel). Le fichier de sortie peut être utilisé par *pvoc* pour générer des fragments audio basés sur le son échantillonné original, avec des échelles de temps et des hauteurs arbitraires et modifiées dynamiquement. L'analyse est conditionnée par les options ci-dessous. Un espace est facultatif entre le drapeau et son argument.

*-s srate* -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur srate de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

*-c canal* -- numéro du canal à traiter. La valeur par défaut est 1.

*-b début* -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

*-d durée* -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie la fin du fichier.

*-n tailletrame* -- taille de trame STFT, le nombre d'échantillons dans chaque trame de l'analyse de Fourier. Doit être une puissance de deux dans l'intervalle 16 à 16384. Pour des résultats propres, une trame doit être plus grande que la période de hauteur la plus longue du son échantillonné. Cependant, des trames très longues donnent un "brouillage" temporel ou une réverbération. La largeur de bande de chaque bin de STFT est déterminée par le rapport *srate / tailletrame*. La taille de trame par défaut est la plus petite puissance de deux qui correspond à plus de 20 ms de la source (par exemple 256 points avec un échantillonnage à 10 kHz, donnant une trame de 25,6 ms).

*-w factfen* -- facteur de chevauchement de fenêtre. Il contrôle le nombre de trames de transformation de Fourier par seconde. *pvoc* interpolera entre les trames, mais un nombre insuffisant de trames générera des distorsions audibles ; trop de trames donneront un fichier d'analyse gigantesque. 4 est un bon com-

promis pour *factfen*, signifiant que chaque point d'entrée apparaît dans 4 fenêtres de sortie, ou inversement que le décalage entre trames de STFT successives est *tailletrame* / 4. La valeur par défaut est 4. N'utilisez pas cette option en même temps que *-h*.

*-h taillesaut* -- décalage de trame STFT. Le contraire de l'option précédente, spécifiant l'incrément en échantillons entre les trames d'analyse successives (voir aussi *lpanal*). N'utilisez pas cette option en même temps que *-w*.

*-H* -- utilise une fenêtre de Hamming à la place de la fenêtre de von Hann employée par défaut.

*-K* -- utilise une fenêtre de Kaiser à la place de la fenêtre de von Hann employée par défaut. Le paramètre de la fenêtre de Kaiser vaut 6,8 par défaut, mais il peut être fixé avec l'option *-B*.

*-B beta* -- fixe le paramètre beta d'une fenêtre de Kaiser utilisée, à la valeur en virgule flottante *beta*.

## Exemples

```
pvanal unson fichierpv
```

analysera le fichier son "unson" en utilisant les valeurs par défaut de *tailletrame* et de *factfen* pour produire le fichier "pvfile" approprié pour une utilisation avec *pvoc*.

## Fichiers

Le fichier de sortie a un en-tête spécial *pvoc* contenant les détails du fichier source audio, le taux des trames d'analyse et le facteur de chevauchement. Les trames de données de l'analyse sont stockées en virgule flottante, avec la magnitude et la « fréquence » (en Hz) des  $N/2 + 1$  premiers bins de Fourier de chaque trame successive. La « fréquence » encode l'incrément de phase de façon à donner une bonne indication de la fréquence réelle pour les harmoniques à fort niveau. Pour les faibles amplitudes ou les harmoniques évoluant rapidement c'est moins significatif.

## Diagnostiques

Imprime le nombre total de trames, et le nombre de trames complétées toutes les 20 trames.

## Crédits

Auteur : Dan Ellis

MIT Media Lab

Cambridge, Massachussetts

1990

## Requêtes sur un Fichier (SNDINFO)

L'utilitaire suivant existe pour les requêtes sur un fichier son :

- *SNDINFO*: Affiche de l'information sur un fichier son.

## sndinfo

sndinfo — Affiche de l'information sur un fichier son.

### Description

Fournit l'information de base sur un ou plusieurs fichiers son.

### Syntaxe

```
csound -U sndinfo [options] fichierson ...
```

```
sndinfo [options] fichierson ...
```

### Initialisation

*sndinfo* tentera de trouver chaque fichier nommé, de l'ouvrir en lecture, de lire l'en-tête du fichier son, pour ensuite imprimer un rapport sur l'information de base trouvée. L'ordre de recherche dans les répertoires de fichiers son est celle qui a été décrite précédemment. Si le fichier est de type AIFF, quelques détails plus avancés sont listés en premier.

Il y a deux types d'options :

1. *-i* ou *-il* imprimera l'information d'instrument, qui comprend les boucles. L'option continue jusqu'à une option *-i0*.
2. L'autre option est *-b* qui imprime l'information de diffusion pour les fichier WAV. Elle peut être arrêtée de façon similaire avec *-b0*.

### Exemples

```
csound -U sndinfo test Bosendorfer/"BOSEN mf A0 st" foo foo2
```

où l'on a les variables d'environnement SFDIR = /u/bv/sound, et SSDIR = /so/bv/Samples, pourra produire ceci :

```
util SNDINFO:
/u/bv/sound/test:
    srate 22050, monaural, 16 bit shorts, 1.10 seconds
    headersiz 1024, datasiz 48500 (24250 sample frames)

/so/bv/Samples/Bosendorfer/BOSEN mf A0 st: AIFF, 197586 stereo samples, base Frq 261.6 (MIDI 60),
AIFF soundfile, looping with modes 1, 0
    srate 44100, stereo, 16 bit shorts, 4.48 seconds

    headersiz 402, datasiz 790344 (197586 sample frames)

/u/bv/sound/foo:
    no recognizable soundfile header

/u/bv/sound/foo2:
    couldn't find
```

## Conversion de Fichier (, HET\_EXPORT, HET\_IMPORT,



## PVLOOK, PV\_EXPORT, PV\_IMPORT, SDIF2AD, SR-CONV)

Les utilitaires suivants existent pour la conversion de fichier :

- *HET\_EXPORT* : Exporte un fichier .het (produit par *HETRO*) vers un fichier texte à séparateur virgule.
- *HET\_IMPORT* : Génère un fichier .het (dans le format produit par *HETRO*) à partir d'un fichier texte à séparateur virgule pour l'utiliser avec le générateur *adsyn*.
- *PVLOOK* : affiche une sortie texte formatée de fichiers d'analyse STFT.
- *PV\_EXPORT* : Convertit un fichier généré par *PVANAL* en un fichier texte.
- *PV\_IMPORT* : Convertit un fichier texte (dans le format généré par *PV\_EXPORT*) en un fichier de format *PVANAL* à utiliser par l'opcode *pvoc*.
- *SDIF2AD* : Convertit des fichiers SDIF en fichiers utilisables par *adsynt*.
- *SRCONV* : Convertit le taux d'échantillonnage d'un fichier audio.

## dnoise

dnoise — Réduit le bruit dans un fichier.

### Description

C'est un schéma de réduction de bruit au moyen du seuillage de bruit dans le domaine fréquentiel.

### Syntaxe

```
dnoise [options] -i ficref_bruit -o ficson_sortie ficson_entree
```

### Initialisation

Options spécifiques à dnoise :

- (*pas d'option*) fichier son en entrée à débruiter
- *-i nomfic* fichier de référence du bruit en entrée
- *-o nomfic* fichier son de sortie
- *-N fnum* nombre de filtres passe-bande (par défaut : 1024)
- *-w fovlp* facteur de chevauchement des filtres : {0,1,(2),3} NE PAS UTILISER *-w* ET *-M*
- *-M longfa* longueur de la fenêtre d'analyse (par défaut : N-1 à moins que *-w* ne soit spécifié)
- *-L longfs* longueur de la fenêtre de synthèse (par défaut : M)
- *-D factd* facteur de décimation (par défaut : M/8)
- *-b datedeb* date de début dans le fichier de référence du bruit (par défaut : 0)
- *-B smpdeb* échantillon de départ dans le fichier de référence du bruit (par défaut : 0)
- *-e datefin* date de fin dans le fichier de référence du bruit (par défaut : fin du fichier)
- *-E smpfin* échantillon de fin dans le fichier de référence du bruit (par défaut : fin du fichier)
- *-t seuil* seuil au-dessus du bruit de référence en dB (par défaut : 30)
- *-S gfact* raideur de la coupure au seuil de bruit, intervalle : 1 à 5 (par défaut : 1)
- *-n nbrtrm* nombre de trames de TFR sur lesquelles calculer la moyenne (par défaut : 5)
- *-m gainmin* gain minimum du seuillage de bruit lorsqu'il est fermé (par défaut : -40)

Options de format du fichier son :

- *-A* format de sortie AIFF
- *-W* format de sortie WAV

- *-J* format de sortie IRCAM
- *-h* pas d'en-tête de fichier (non valide pour une sortie AIFF/WAV)
- *-8* échantillons en caractères non signés sur 8 bit
- *-c* échantillons en caractères signés sur 8 bit
- *-a* échantillons en alaw
- *-u* échantillons en ulaw
- *-s* échantillons en entiers courts
- *-l* échantillons en entiers longs
- *-f* échantillons en virgule flottante. Les nombres en virgule flottante sont aussi supportés par les fichiers WAV. (Nouveau dans Csound 3.47.)

Options supplémentaires :

- *-R* verbose - impression d'une information d'état
- *-H [N]* imprime un caractère de type pulsation à chaque écriture dans le fichier son.
- *-- nomfic* sortie de journal dans le fichier nomfic
- *-V* verbose - impression d'une information d'état



### Note

DNOISE consulte aussi la variable d'environnement SFOUTYP pour déterminer le format du fichier de sortie.

L'option *-i* est utilisée pour un fichier de référence du bruit (créé normalement à partir d'un court extrait du fichier à débiter, dans lequel seul le bruit est audible). Le fichier son d'entrée à débiter peut être donné n'importe où dans la ligne de commande, sans drapeau.

## Exécution

C'est un schéma de réduction de bruit au moyen du seuillage de bruit dans le domaine fréquentiel. Il fonctionnera mieux dans le cas d'un rapport signal/bruit élevé avec un bruit de type souffle.

L'algorithme est celui suggéré par Moorer & Berger dans « Linear-Phase Bandsplitting: Theory and Applications » présenté à la 76ème Convention, 8-11 Octobre 1984 à New York, de l'Audio Engineering Society (préimpression #2132) sauf qu'il utilise la formulation par Chevauchement-Addition Pondérée pour l'analyse et la synthèse de Fourier à court terme au lieu de la formulation récursive proposée par Moorer & Berger. Le gain pour chaque bin de fréquence est calculé indépendamment selon la formule

$$\text{gain} = g0 + (1-g0) * [\text{moy} / (\text{moy} + \text{th} * \text{th} * \text{nref})] ^ \text{sh}$$

où *moy* et *nref* sont la moyenne quadratique du signal et du bruit respectivement pour le bin en question. (Ceci diffère légèrement de la version dans Moorer & Berger.)

Les paramètres critiques *th* et *g0* sont spécifiés en dB et convertis en interne en valeurs décimales. Les

valeurs *nref* sont calculées au début du programme sur la base d'un fichier de bruit (spécifié dans la ligne de commande) qui contient du bruit sans signal.

Les valeurs *moy* sont calculée sur une fenêtre rectangulaire de *m* trames de TFR centrée sur la date courante. Cela correspond à une extension temporelle de  $m \cdot D/R$  (qui vaut typiquement  $(m \cdot N/8)/R$ ). Le réglage par défaut de *N*, *m*, et *D* devrait convenir pour la plupart des utilisations. Un taux d'échantillonnage supérieur à 16 kHz pourrait signifier un *N* plus grand.

## Crédits

Auteur : Mark Dolson

26 août 1989

Auteur : John ffitch

30 décembre 2000

Mis à jour par Rasmus Ekman le 11 mars 2002.

## het\_export

het\_export — Convertit un fichier .het en fichier texte à séparateur virgule.

### Syntaxe

```
het_export fichier_het fichier_textecsv
```

```
csound -U het_export fichier_het fichier_textecsv
```

### Initialisation

*fichier\_het* - Nom du fichier d'entrée .het.

*fichier\_textecsv* - Nom du fichier texte à séparateur virgule.

L'utilitaire *het\_export* génère un fichier texte à séparateur virgule pour pouvoir éditer manuellement un fichier .het produit par l'utilitaire *HETRO*. On peut l'utiliser en combinaison avec *het\_import* pour produire des données pour le générateur *adsyn*.

### Crédits

Auteur : John ffitch

1995

## het\_import

het\_import — Convertit un fichier texte à séparateur virgule en un fichier .het

### Syntaxe

```
het_import fichier_textecsv fichier_het
```

```
csound -U het_import fichier_textecsv fichier_het
```

### Initialisation

*fichier\_textecsv* - Nom du fichier texte à séparateur virgule.

*fichier\_het* - Nom du fichier .het de sortie.

L'utilitaire *het\_import* génère un fichier *.het* utilisable avec le générateur *adsyn*. Il peut être utilisé en combinaison avec *het\_export* pour modifier l'analyse du son faite par l'utilitaire *HETRO*.

### Crédits

Auteur : John ffitch

1995

pvlook — Affiche une sortie texte formatée de fichiers d'analyse STFT.

Affiche une sortie texte formatée de fichiers d'analyse STFT créés avec *pvanal*.

**csound** -U **pvlook** [options] fichier\_entree

**pvlook** [options] fichier entree

*pvlook* lit un fichier, et les trajectoires de fréquence et d'amplitude pour chacun des bins de l'analyse, dans un format texte lisible. Le fichier est supposé être un fichier d'analyse STFT créée par *pvanal*. Par défaut, le fichier entier est traité.

**-bb n** -- commence au bin d'analyse numéro *n*, numérotés à partir de 1. La valeur par défaut est 1.

-eb *n* -- termine au bin d'analyse numéro *n*. Vaut par défaut la valeur la plus haute.

*-bf n* -- commence à la trame d'analyse numéro *n*, numérotées à partir de 1. La valeur par défaut est 1.

-ef *n* -- termine à la trame d'analyse numéro *n*. Vaut par défaut la valeur la plus haute.

*-i* -- imprime les valeurs en entier. Par défaut en virgule flottante.

[illegible]





[illegible]

2475

```

0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.140 1.265 2.766 3.289 3.296 3.293 3.296 3.296 3.290 3.293
3.292 3.291 3.297 3.295 3.294 3.296 3.291 3.292 3.294 3.291
3.296 3.297 3.292 3.295 3.292 3.290 3.295 3.293 3.294 3.297
3.292 3.293 3.294 3.290 3.295 3.295 3.292 3.296 3.293 3.291
3.294 3.291 3.293 3.297 3.292 3.295 3.294 3.288 3.293 3.293
3.292 3.297 3.294 3.292 3.295 3.290 3.292 3.295 3.292 3.295
3.295 3.290 3.294 3.292 3.292 3.297 3.293 3.293 3.295 3.290
3.292 3.293 3.290 3.296 3.296 3.292 3.295 3.291 3.290 3.294
3.291 3.294 3.296 3.291 3.293 3.293 3.290 3.295 3.294 3.293
3.296 3.291 3.291 3.293 3.293 3.290 3.294 3.296 3.292 3.295 3.293
3.288 3.293 3.292 3.292 3.297 3.292 3.293 3.294 3.289 3.292
3.294 3.291 3.296 3.293 3.291 3.294 3.291 3.292 3.296 3.292
3.294 3.295 3.289 3.292 3.292 3.291 3.296 3.294 3.292 3.295
3.290 3.290 3.293 3.291 3.295 3.296 3.291 3.294 3.291 3.289
3.294 3.292 3.293 3.295 3.291 3.292 3.293 3.290 3.294 3.295
3.292 3.294 3.291 3.289 3.293 3.291 3.293 3.296 3.292 3.293
3.293 3.288 3.292 3.293 3.292 3.296 3.293 3.291 3.294 3.289
3.292 3.295 3.291 3.294 3.293 3.289 3.292 3.291 3.290 3.295
3.293 3.292 3.294 3.289 3.291 3.293 3.295 3.290 3.292 3.292
3.289 3.293 3.293 3.292 3.295 3.291 3.289 3.292 3.290 3.292
3.295 3.291 3.293 3.292 3.288 3.292 3.291 3.291 3.295 3.291
3.291 3.292 3.289 3.291 3.294 3.291 3.294 3.292 3.289 3.292
3.290 3.290 3.295 3.292 3.293 3.294 3.289 3.291 3.292 3.290
3.294 3.293 3.291 3.293 3.289 3.290 3.293 3.291 3.294 3.295
3.290 3.292 3.291 3.289 3.294 3.293 3.292 3.294 3.290 3.290
3.292 3.289 3.293 3.294 3.291 3.293 3.291 3.289 3.292 3.291
3.291 3.295 3.291 3.291 3.292 3.288 3.292 3.293 3.291 3.295
3.292 3.290 3.292 3.289 3.291 3.294 3.291 3.293 3.292 3.288
3.291 3.291 3.290 3.295 3.292 3.291 3.293 3.289 3.290 3.292
3.290 3.294 3.293 3.290 3.292 3.290 3.289 3.293 3.291 3.292
3.294 3.290 3.290 3.291 3.289 3.293 3.293 3.291 3.293 3.290
3.288 3.291 3.290 3.292 3.292 3.294 3.290 3.292 3.291 3.288 3.291
3.291 3.291 3.294 3.291 3.290 3.291 3.288 3.291 3.293 3.291
3.293 3.292 3.288 3.291 3.290 3.290 3.294 3.291 3.291 3.292
3.288 3.290 3.291 3.290 3.294 3.293 3.290 3.292 3.289 3.289
3.293 3.290 3.292 3.293 3.289 3.291 3.290 3.289 3.293 3.292
3.291 3.293 3.289 3.289 3.291 3.289 3.292 3.293 3.290 3.292
3.290 3.288 3.292 3.291 3.291 3.294 3.290 3.290 3.291 3.288
3.291 3.292 3.291 3.293 3.291 3.288 3.291 3.289 3.290 3.293
3.290 3.292 3.292 3.288 3.291 3.291 3.290 3.293 3.291 3.290
3.292 3.288 3.292 3.292 3.290 3.292 3.293 3.289 3.291 3.289
3.288 3.293 3.291 3.291 3.292 3.288 3.289 3.290 3.288 3.292
3.293 3.290 3.292 3.289 3.288 3.291 3.290 3.291 3.293 3.289
3.290 3.290 3.287 3.291 3.291 3.290 3.293 3.290 3.288 3.290
3.288 3.290 3.293 3.291 3.292 3.291 3.288 3.290 3.289 3.289
3.293 3.290 3.290 3.291 3.287 3.289 3.291 3.289 3.292 3.291
3.288 3.290 3.288 3.288 3.292 3.290 3.291 3.292 3.288 3.289
3.290 3.288 3.292 3.292 3.290 3.292 3.289 3.288 3.291 3.289
3.291 3.293 3.289 3.291 3.290 3.287 3.291 3.290 3.290 3.293
3.289 3.289 3.290 3.287 3.290 3.292 3.290 3.292 3.290 3.287
3.290 3.289 3.289 3.292 3.290 3.290 3.291 3.287 3.289 3.290
3.289 3.292 3.291 3.289 3.291 3.288

```

etc...

## Crédits

Auteur : Richard Karpen

Seattle, Wash

1993 (Nouveau dans la version 3.57 de Csound)

## **pv\_export**

`pv_export` — Convertit un fichier .pvx en fichier texte à séparateur virgule.

### **Syntaxe**

```
pv_export fichier_pv fichier_texte_csv
```

```
csound -U pv_export fichier_pv fichier_texte_csv
```

### **Initialisation**

*fichier\_pv* - Nom du fichier d'entrée .pvx.

*fichier\_texte\_csv* - Nom du fichier texte à séparateur virgule de sortie.

L'utilitaire *pv\_export* génère un fichier texte à séparateur virgule pour une édition manuelle d'un fichier .pvx produit par l'utilitaire *PVANAL*. Il peut être utilisé en combinaison avec *pv\_import* pour produire des données pour le générateur *pvoc*.

### **Crédits**

Auteur : John ffitch

1995

## **pv\_import**

`pv_import` — Convertit un fichier texte à séparateur virgule en un fichier .pvx.

### **Syntaxe**

```
pv_import fichier_texte_csv fichier_pv
```

```
csound -U pv_import fichier_texte_csv fichier_pv
```

### **Initialisation**

*fichier\_texte\_csv* - Nom du fichier texte à séparateur virgule en entrée.

*fichier\_pv* - Nom du fichier .pvx de sortie.

L'utilitaire *pv\_import* génère un fichier .pvx utilisable avec le générateur *pvoc*. Il peut être utilisé en combinaison avec *pv\_export* pour modifier une analyse de son faite par l'utilitaire *PVANAL*.

### **Crédits**

Auteur : John ffitch

1995

## sdif2ad

sdif2ad — Convertit des fichiers SDIF en fichiers utilisables par adsynt.

### Description

Convertit des fichiers Sound Description Interchange Format (SDIF) dans le format utilisable par l'opcode de Csound *adsyn*. A partir de la version 4.10 de Csound, *sdif2ad* n'est plus disponible que comme un programme autonome pour console Windows et pour DOS.

### Syntaxe

```
sdif2ad [options] fichier_entree fichier_sortie
```

### Initialisation

Options :

- *-sN* -- applique le facteur d'échelle d'amplitude N
- *-pN* -- ne garde que les N premiers partiels. Limité à 1024 partiels. Les indices de piste de partiels de la source sont utilisés directement pour sélectionner le stockage interne. Comme ils peuvent avoir des valeurs arbitraires, le maximum de 1024 partiels peut ne pas être réalisé dans tous les cas.
- *-r* -- fichier de données de sortie en octets inversés. L'option octets inversés est là pour faciliter le transfert entre plates-formes, car le format de fichier *adsyn* de Csound n'est pas portable.

Si le nom de fichier passé à *hetro* a l'extension « .sdif », les données seront écrites en format SDIF comme des trames 1TRC de données de synthèse additive. Le programme utilitaire *sdif2ad* peut être utilisé pour convertir tout fichier SDIF contenant un flot de données 1TRC dans le format *adsyn* de Csound. *sdif2ad* permet à l'utilisateur de limiter le nombre de partiels retenus, et d'appliquer un facteur d'échelle d'amplitude. Ceci est souvent nécessaire, car la spécification SDIF, depuis la réalisation de *sdif2ad*, ne nécessite pas que les amplitudes soient dans un intervalle particulier. *sdif2ad* rapporte sur la console l'information sur le fichier, y compris l'intervalle de fréquence.

Les principaux avantages de SDIF sur le format *adsyn*, pour les utilisateurs de Csound, sont que les fichiers SDIF sont totalement portables d'une plate-forme à l'autre (les données sont en « big-endian »), et qu'ils n'ont pas la limite de durée de 32,76 secondes imposée par le format *adsyn* sur 16 bit. Cette limite est nécessairement imposée par *sdif2ad*. Dans le futur, la lecture du format SDIF pourra être incorporée directement dans *adsyn*, permettant ainsi l'analyse et le traitement de fichiers de n'importe quelle longueur (seulement limitée par la capacité mémoire du système).

Les utilisateurs doivent se souvenir que les formats SDIF sont toujours en développement. Bien que le format 1TRC soit maintenant bien établi, il peut encore changer.

Pour des informations détaillées sur le Sound Description Interchange Format, se référer au site web du CNMAT : <http://cnmat.CNMAT.Berkeley.EDU/SDIF>

D'autres ressources SDIF (y compris un visionneur) sont disponibles via le site web de NC\_DREAM : <http://www.bath.ac.uk/~masjpf/NCD/dreamhome.html>

### Crédits

Auteur : Richard Dobson

Somerset, England

Août 2000

Nouveau dans la version 4.08 de Csound

## srconv

srconv — Convertit le taux d'échantillonnage d'un fichier audio.

### Description

Convertit le taux d'échantillonnage d'un fichier audio de *Rin* à *Rout*. Optionnellement le rapport (*Rin* / *Rout*) peut varier linéairement dans le temps selon un ensemble de paires (temps, rapport) dans un fichier auxiliaire.

### Syntaxe

```
srconv [options] fichier_entree
```

### Initialisation

Options :

- *-P num* = rapport de transposition en hauteur (*srate* / *r*) [ne pas spécifier à la fois *P* et *r*]
- *-Q num* = facteur de qualité (1, 2, 3 ou 4 : par défaut = 2)
- *-i nomfic* = fichier auxiliaire de points charnière (pas de point charnière par défaut, c'est-à-dire pas de changement de rapport)
- *-r num* = taux d'échantillonnage en sortie (doit être spécifié)
- *-o nomfic* = nom du fichier son de sortie
- *-A* = crée un fichier son de sortie au format AIFF
- *-J* = crée un fichier son de sortie au format IRCAM
- *-W* = crée un fichier son de sortie au format WAV
- *-h* = pas d'en-tête dans le fichier son de sortie
- *-c* = échantillons en caractères signés sur 8 bit
- *-a* = échantillons alaw
- *-8* = échantillons en caractères non-signés sur 8 bit
- *-u* = échantillons ulaw
- *-s* = échantillons en entiers courts
- *-l* = échantillons en entiers longs
- *-f* = échantillons en virgule flottante
- *-r N* = remplace le *srate* de l'orchestre
- *-K* = ne génère pas de bloc de pics d'amplitude
- *-R* = réécrit continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF)

- *-H#* = imprime une pulsation dans le style 1, 2 ou 3 à chaque écriture dans le fichier son
- *-N* = notification (cloche système) quand le traitement est fini
- *-- nomfic* = compte-rendu dans un fichier

Ce programme effectue une conversion arbitraire du taux d'échantillonnage en haute fidélité. La méthode consiste à parcourir le fichier d'entrée avec un pas d'incrément conforme au taux d'échantillonnage désiré, et de calculer les points de sortie comme moyennes convenablement pondérées des points voisins. Il y a deux cas à considérer :

1. les taux d'échantillonnage sont dans un petit rapport entier - les poids sont obtenus de la table
2. les taux d'échantillonnage sont dans un grand rapport entier - les poids sont linéairement interpolés de la table.

Calcul de l'incrément : pour une décimation, la fenêtre est la réponse impulsionnelle d'un filtre passe-bas avec une fréquence de coupure située à la moitié de la fréquence d'échantillonnage en sortie ; pour une interpolation, la fenêtre est la réponse impulsionnelle d'un filtre passe-bas avec une fréquence de coupure située à la moitié de la fréquence d'échantillonnage de l'entrée.

## Crédits

Auteur : Mark Dolson

26 août 1989

Auteur : John ffitich

30 décembre 2000

## Autres Utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MAKECSD, MIXER, SCALE)

Les divers utilitaires suivants sont disponibles :

- *CS* : Démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.
- *CSB64ENC* : Convertit un fichier binaire en un fichier texte encodé en Base64.
- *ENVEXT* : Extrait l'enveloppe d'un fichier vers une liste textuelle.
- *EXTRACTOR* : Extrait une section audio d'un fichier audio.
- *MAKECSD* : Crée un fichier CSD à partir des fichiers d'entrée spécifiés.
- *MIXER* : Mélange ensemble plusieurs fichiers son.
- *SCALE* : Calibre l'amplitude d'un fichier son.



## CS

**cs** — Démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.

## Description

Démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.

## Syntaxe

```
cs [-OPTIONS] <nom> [OPTIONS DE CSOUND ... ]
```

## Initialisation

Drapeaux :

- - *OPTIONS* = *OPTIONS* est une séquence de caractères alphabétiques qui peut être utilisée pour sélectionner l'exécutable Csound à lancer, aussi bien que les options de ligne de commande (voir ci-dessous). L'option 'r' est une valeur par défaut (sélection de la sortie en temps-réel), mais on peut la remplacer.
- <nom> = c'est la racine de nom de fichier pour sélectionner les fichiers arguments ; elle peut contenir un chemin. Les fichiers qui ont pour extension .csd, .orc, ou .sco sont recherchés, et soit un CSD soit une paire orc/sco qui correspond à <nom>, le meilleur des deux, est sélectionné. Des fichiers MIDI avec une extension .mid sont aussi recherchés, et si l'un d'eux correspond à <nom> au moins autant que le CSD ou la paire orc/sco, il est utilisé avec l'option -F.



### NOTE

Le fichier MIDI n'est pas utilisé si une option -M ou -F est spécifiée par l'utilisateur (nouveau dans la version 4.24.0). A moins qu'il y ait une option (-n ou -o) relative à la sortie audio, un nom de fichier de sortie avec l'extension appropriée est généré automatiquement (basé sur le nom des fichiers d'entrée sélectionnés et sur les options de format). Le fichier de sortie est toujours écrit dans le répertoire courant.



### NOTE

les extensions de nom de fichier ne sont pas sensibles à la casse.

- [*OPTIONS DE CSOUND ...*] = n'importe quel nombre d'options supplémentaires pour Csound qui sont simplement copiées dans la ligne de commande finale qui sera exécutée.

La ligne de commande qui est exécutée est générée à partir de quatre origines :

1. L'exécutable de Csound (éventuellement avec options). Une seule possibilité est choisie parmi les trois qui suivent (la dernière à la plus haute priorité) :
  - une valeur par défaut

- la valeur d'une variable d'environnement de CSOUND
  - des variables d'environnement avec un nom de la forme CSOUND\_x où x est une lettre majuscule choisie parmi les caractères de la chaîne -OPTIONS. Ainsi, si l'option -dcbA est utilisée, et si les variables d'environnement CSOUND\_B et CSOUND\_C sont définies, la valeur de CSOUND\_B sera effective.
2. N'importe quel nombre de listes d'option, ajoutées dans l'ordre suivant :
- soit quelques valeurs par défaut, soit la valeur de la variable d'environnement CSFLAGS si elle est définie.
  - des variables d'environnement avec un nom de la forme CSFLAGS\_x où x est une lettre majuscule choisie parmi les caractères de la chaîne -OPTIONS. Ainsi, si l'option -dcbA est utilisée, et si les variables d'environnement CSFLAGS\_A et CSFLAGS\_C sont définies par '-M 1 -o dac' et '-m231 -H0', respectivement, la chaîne '-m231 -H0 -M 1 -o dac' sera ajoutée.
3. Les options explicites de [OPTIONS DE CSOUND ... ].
4. Toutes les options et les noms de fichiers générés à partir de <nom>.



## NOTE

Les options entre apostrophes qui contiennent des espaces sont autorisées.

## Exemples

Avec les variables d'environnement suivantes :

```
CSOUND      = csoundfltk.exe -W
CSOUND_D    = csound64.exe -J
CSOUND_R    = csoundfltk.exe -h

CSFLAGS     = -d -m135 -H1 -s
CSFLAGS_D   = -f
CSFLAGS_R   = -m0 -H0 -o dac1 -M "MIDI Yoke NT: 1" -b 200 -B 6000
```

Et un répertoire qui contient :

```
foo.orc      piano.csd
foo.sco      piano.mid
im.csd       piano2.mid
ImproSculpt2_share.csd foobar.csd
```

Les commandes suivantes s'exécuteront comme il est montré :

```
cs foo      => csoundfltk.exe -W -d -m135 -H1 -s -o foo.wav \
foo.orc foo.sco

cs foob     => csoundfltk.exe -W -d -m135 -H1 -s          \
-o foobar.wav foobar.csd

cs -r imp -i adc => csoundfltk.exe -h -d -m135 -H1 -s -m0 -H0 \
-o dac1 -M "MIDI Yoke NT: 1" \
-b 200 -B 6000 -i adc \
ImproSculpt2_share.csd

cs -d im     => csound64.exe -J -d -m135 -H1 -s -f -o im.sf \
im.csd

cs piano    => csoundfltk.exe -W -d -m135 -H1 -s          \
-F piano.mid -o piano.wav \
```

```
piano.csd  
cs piano2          => csoundfltk.exe -W -d -m135 -H1 -s      \  
-F piano2.mid -o piano2.wav      \  
piano.csd
```

## Crédits

Auteur : Istvan Varga

Janvier 2003

## csb64enc

csb64enc — Convertit un fichier binaire en un fichier texte encodé en Base64.

### Description

L'utilitaire *csb64enc* génère un fichier texte encodé en Base64 à partir d'un fichier binaire, tel qu'un fichier MIDI standard (.mid) ou n'importe quel type de fichier audio. Il est utile pour convertir un fichier dans le format accepté par la section *<CsFileB>* d'un fichier csd, pour y inclure le fichier converti.

### Syntaxe

```
csb64enc [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]
```

### Initialisation

Options :

- - *w n* = fixe la largeur de ligne du fichier de sortie à *n* (par défaut : 72)
- - *o nomfic* = nom du fichier de sortie (par défaut : stdout)

### Exemples

```
csb64enc -w 78 -o fichier.txt fichier.mid
```

La commande produit un fichier texte encodé en Base64 à partir d'un fichier MIDI standard, *fichier.mid*. Ce fichier peut maintenant être collé dans la section *<CsFileB>* d'un fichier csd.

### Voir Aussi

*makecsd*

### Crédits

Auteur : Istvan Varga

Janvier 2003

## envext

envext — Extrait l'enveloppe d'un fichier son vers un fichier texte.

## Syntaxe

```
envext [-options] fichierson
```

```
csound -U envext [-options] fichierson
```

## Initialisation

*fichierson* - Nom du fichier son en entrée.

Les options suivantes sont disponibles pour *envext*. (Les valeurs par défaut sont mises entre parenthèses) :

-o *nomfic* Nom du fichier de sortie (newenv)

-w *taille* (en secondes) de la fenêtre d'analyse (0.25)

L'utilitaire *envext* génère un fichier texte contenant des paires de temps et d'amplitude en trouvant les pics absolus dans chaque fenêtre.

## Exemple

En tapant la commande (depuis le répertoire manual-fr) :

```
csound -U envext examples/mary.wav
```

on obtiendra un fichier texte contenant :

```
0.000 0.000
0.000 0.000
0.250 0.000
0.500 0.000
0.750 0.000
1.249 0.170
1.499 0.269
1.530 0.307
1.872 0.263
2.056 0.304
2.294 0.241
2.570 0.216
2.761 0.178
3.077 0.011
3.251 0.001
3.500 0.000
```

qui montre le temps pour le pic d'amplitude dans chaque fenêtre mesurée.

## Crédits

Auteur : John ffitch

1995

## extractor

extractor — Extrait une section audio d'un fichier audio.

### Description

Extrait une section audio, par temps ou échantillon, d'un fichier son existant.

### Syntaxe

```
extractor [OPTIONS ... ] fichierentree
```

### Initialisation

Options :

- *-S entier* = Démarre l'extraction à l'échantillon dont le numéro est donné.
- *-Z entier* = Termine l'extraction à l'échantillon dont le numéro est donné.
- *-Q entier* = Extrait le nombre donné d'échantillons.
- *-T fpnum* = Démarre l'extraction au temps donné en secondes.
- *-E fpnum* = Termine l'extraction au temps donné en secondes.
- *-D fpnum* = Extrait la durée donnée en secondes.
- *-R* = Réécrit continuellement l'en-tête lors de l'écriture du fichier son (WAV/AIFF).
- *-H entier* = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.
- *-N* = Signal d'alerte (habituellement la cloche système) à la fin.
- *-v* = Mode verbeux.
- *-o nomfic* = Nom du fichier de sortie (par défaut : test.wav)

### Exemples

Les valeurs par défaut sont :

```
extractor -S 0 -Z fin-du-fichier -o test
```

Par exemple

```
extractor -S 10234 -D 2.13 in.aiff -o out.wav
```

Cela crée un nouveau fichier son extrait à partie de l'échantillon 10234 et durant 2,13 secondes.

### Crédits

Auteur : John ffitich

1994

## makecsd

makecsd — Crée un fichier CSD à partir des fichiers spécifiés en entrée.

### Description

Crée un fichier CSD à partir des fichiers spécifiés en entrée. Le premier fichier d'entrée qui a une extension .orc (la casse n'est pas significative) est mis dans la section <CsInstruments>, et le premier fichier d'entrée qui a une extension .sco devient <CsScore>. Tous les fichiers restants sont encodés en Base64 et ajoutés dans des balises <CsFileB>. Une section <CsOptions> vide est toujours ajoutée.

Un filtrage du texte est effectué sur les fichiers d'orchestre et de partition :

- les caractères de nouvelle ligne sont convertis dans le format natif du système sur lequel *makecsd* est exécuté.
- les lignes vides sont enlevées du début et de la fin des fichiers.
- tous les espaces restant en fin de ligne sont supprimés.
- facultativement, les tabulations peuvent être développées en espaces avec une taille de tabulation spécifiée par l'utilisateur.

### Syntaxe

```
makecsd [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]
```

### Initialisation

Options :

- - *t n* = développe les tabulations en espaces en utilisant une taille de tabulation égale à *n* (désactivé par défaut). Ceci s'applique seulement à l'orchestre et à la partition.
- - *w n* = fixe la largeur de ligne Base64 à *n* (par défaut : 72). Note : l'orchestre et la partition ne sont pas concernés.
- - *o nomfic* = nom du fichier de sortie (par défaut : stdout)

### Exemples

```
makecsd -t 6 -w 78 -o fichier.csd fichier.mid fichier.orc fichier.sco sample.aif
```

Crée un fichier CSD à partir de fichier.orc et de fichier.sco (les tabulations sont développées en espaces sachant qu'une tabulation vaut 6 caractères), et fichier.mid et sample.aif sont ajoutés dans des balises <CsFileB> contenant les données encodées en Base64 avec une largeur de ligne de 78 caractères. Le fichier de sortie est fichier.csd.

### Crédits

Auteur : Istvan Varga

Janvier 2003



## mixer

mixer — Mélange ensemble plusieurs fichiers son.

### Description

Mélange ensemble plusieurs fichiers son, démarrant à des temps différents et avec une sélection individuelle des canaux dans les fichiers d'entrée.

### Syntaxe

```
mixer [OPTIONS ... ] fichier [[OPTIONS... ] fichier] ...
```

### Initialisation

Options :

- *-A* = Génère un fichier de sortie en AIFF.
- *-W* = Génère un fichier de sortie en WAV.
- *-h* = Génère un fichier de sortie sans en-tête.
- *-c* = Génère des échantillons en caractères signés sur 8 bit.
- *-a* = Génère des échantillons alaw.
- *-u* = Génère des échantillons ulaw.
- *-s* = Génère des échantillons en entiers courts.
- *-l* = Génère des échantillons en entiers longs (32 bit).
- *-f* = Génère des échantillons en virgule flottante.
- *-F arg* = Spécifie le gain à appliquer au fichier d'entrée qui suit. Si *arg* est un nombre en virgule flottante ce gain est appliqué uniformément à l'entrée. Alternativement ça peut être un nom de fichier qui spécifie un fichier de points charnière pour varier le gain sur différentes périodes.
- *-S entier* = Indique à partir de quel échantillon commencer le mixage dans le fichier d'entrée suivant.
- *-T fpnum* = Indique à quel date (en secondes) commencer le mixage dans le fichier d'entrée suivant.
- *-1* = Mixer le canal 1 du fichier son suivant.
- *-2* = Mixer le canal 2 du fichier son suivant.
- *-3* = Mixer le canal 3 du fichier son suivant.
- *-4* = Mixer le canal 4 du fichier son suivant.
- *-^ entx enty* = Mixer le canal *x* du fichier son suivant vers le canal *y* dans le fichier de sortie.
- *-v* = Mode verbeux.
- *-R* = Réécrit continuellement l'en-tête lors des opérations d'écriture du fichier son (WAV/AIFF)

- *-H entier* = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.
- *-N* = Alerte (habituellement la cloche système) lorsque le mixage est fini.
- *-o nomfic* = nom du fichier de sortie (par défaut : test.wav)

## Exemples

Les valeurs par défaut sont :

```
mixer -s -otest -F 1.0 -S 0
```

Par exemple

```
mixer -F 0.96 in1.wav -S 300 -2 in2.aiff -S 300 -^4 1 in3.wav -o out.wav
```

Cela crée un nouveau fichier son avec un gain constant de 0,96 pour in1.wav, le second canal de in2.aiff mixé après 300 échantillons et le canal 4 de in3.wav sorti comme le canal 1 après 300 échantillons.

## Crédits

Auteur : John ffitch

1994

## scale

scale — Calibre l'amplitude d'un fichier son.

### Description

Prend un fichier son et le calibre en appliquant un gain, constant ou variable. L'échelle peut être comme un multiplicateur, un maximum ou un pourcentage de 0dB.

### Syntaxe

```
scale [OPTIONS ... ] fichier
```

### Initialisation

Options :

- *-A* = Génère un fichier de sortie AIFF.
- *-W* = Génère un fichier de sortie WAV.
- *-h* = Génère un fichier de sortie sans en-tête.
- *-c* = Génère des échantillons en caractères signés sur 8 bit.
- *-a* = Génère des échantillons alaw.
- *-u* = Génère des échantillons ulaw.
- *-s* = Génère des échantillons en entiers courts.
- *-l* = Génère des échantillons en entiers longs (32 bit)
- *-f* = Génère des échantillons en virgule flottante.
- *-F arg* = Spécifie le gain à appliquer. Si *arg* est un nombre en virgule flottante ce gain est appliqué uniformément à l'entrée. Alternativement ça peut être un nom de fichier qui spécifie un fichier de points charnière pour varier le gain sur différentes périodes.
- *-M fpnum* = Calibre l'entrée de façon telle que la valeur absolue du déplacement maximum soit la valeur donnée.
- *-P fpnum* = Calibre l'entrée de façon telle que la valeur absolue du déplacement maximum soit le pourcentage donné de 0dB.
- *-R* = Réécrit continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).
- *-H entier* = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.
- *-N* = Alerte (habituellement la cloche système) lorsque c'est fini.
- *-o nomfic* = nom du fichier de sortie (par défaut : test.wav)

### Exemples

```
scale -s -W -F 0.96 -o out.wav sound.wav
```

Ceci crée un nouveau fichier son avec un gain constant de 0,96. C'est particulièrement utile si le fichier d'entrée est en format virgule flottante.

## Crédits

Auteur : John ffitch

1994

## Crédits

Dan Ellis

MIT Media Lab

Cambridge, Massachussetts

---

# Cscore

*Cscore* est une API (interface de programmation d'application) pour générer et manipuler des fichiers de partition numérique. Elle fait partie de l'API plus grande de Csound et elle comprend un certain nombre de fonctions appelables depuis un programme écrit par l'utilisateur en langage C. *Cscore* peut être invoquée comme un préprocesseur de partition autonome ou comme élément d'une exécution de Csound en incluant l'option -C dans ses arguments :

```
cscore [fichier_partition_entree] [> fichier_partition_sortie]
```

(où *cscore* est le nom du programme que vous avez écrit), ou

```
csound [-C] [autresoptions] [nomorch] [nompartition]
```

Les fonctions de l'API disponibles augmentent la bibliothèque de fonctions du langage C ; elles peuvent lire des fichiers de partition numérique standard ou pré-triée, modifier et étendre les données de différentes manières, et ensuite les rendre disponibles pour une exécution par un orchestre de Csound.

Le programme écrit par l'utilisateur dans le langage C est compilé et lié à la bibliothèque de Csound (ou au programme de ligne de commande *csound*) par l'utilisateur. Il n'est pas indispensable de bien connaître le langage C pour écrire ce programme, car les appels de fonction ont une syntaxe simple, et sont suffisamment puissants pour faire la plus grande partie du travail compliqué. C pourra apporter plus de puissance par la suite selon les besoins.

Les sections suivantes expliquent toutes les étapes de l'utilisation de *Cscore* :

- *Evènements, Listes et Opérations* - Explique la syntaxe des fonctions de *Cscore* et les structures de données.
- *Ecrire un Programme de Contrôle Cscore* - Montre par l'exemple comment écrire votre propre programme de contrôle.
- *Compiler un Programme Cscore* - Décrit les étapes de la compilation et de l'édition des liens avec la bibliothèque de Csound.
- *Exemples Plus Avancés* - Traite de questions avancées comme plusieurs partitions en entrée et les détails de l'exécution de *Cscore* au sein d'une exécution de Csound.

## Evènements, Listes et Opérations

Un évènement dans *Cscore* est équivalent à une instruction d'une *partition numérique standard* ou d'une partition résolue en temps (le format dans lequel Csound écrit une partition triée -- consultez n'importe quel fichier *score.srt*), et il est stocké en interne en format de temps résolu. Il est important de noter que lorsque *Cscore* est utilisé en mode autonome, il est incapable de comprendre les « commodités » non numériques que Csound permet dans le format de partition en entrée. C'est pourquoi, les partitions utilisant des fonctionnalités telles que le report (carry), les rampes, les expressions et autres devront être triées au préalable avec l'utilitaire *scsort* ou bien utilisées avec un exécutable *Csound* modifié contenant le programme *Cscore* de l'utilisateur. Les opcodes de partition avec des argument macros (r, m, n, and { }) ne sont pas interprétés.

Les événements de partition sont lus à partir d'un fichier de partition existant et stockés chacun dans une structure C. Les principaux composants de ces structures sont un opcode et un tableau de valeurs de p-champs. *Cscore* gère la lecture des événements et leur mise en mémoire pour vous. Le format de la structure commence comme suit :

```
typedef struct {
    CSHDR h;          /* en-tête pour la gestion de l'espace */
    char *strarg;      /* adresse d'un argument chaîne facultatif */
    char op;           /* opcode-t, w, f, i, a, s ou e */
    short pcnt;
    MYFLT p2orig;      /* p2, p3 non résolus */
    MYFLT p3orig;
    MYFLT p[11];       /* tableau des p-champs p0, p1, p2 ... */
} EVENT;
```

MYFLT est l'un des types C *float* ou *double* selon la manière dont votre copie de la bibliothèque de Csound a été compilée. Vous avez juste à déclarer les variables en virgule flottante de votre programme avec le type MYFLT pour être compatible.

Toute fonction de *Cscore* qui crée, lit ou copie un événement retournera un pointeur sur la structure dans laquelle les données de l'événement sont stockées. Ce pointeur d'événement peut être utilisé pour accéder aux composants de la structure, de la forme *e->op* ou *e->p[n]*. Chaque événement nouvellement stocké provoquera la création d'un nouveau pointeur, et une séquence de nouveaux événements générera une séquence de pointeurs distincts qu'il faudra stocker. Les groupes de pointeurs d'événement sont stockés dans une liste d'événements qui a sa propre structure :

```
typedef struct {
    CSHDR h;
    int nslots;        /* nombre maximal d'événements dans cette liste */
    int nevents;       /* nombre d'événements présents */
    EVENT *e[11];      /* tableau de pointeurs d'événement e0, e1, e2.. */
} EVLIST;
```

Toute fonction qui crée ou modifie une liste retournera un pointeur sur la nouvelle liste. Ce pointeur de liste peut être utilisé pour accéder à ses composants pointeurs d'événement, de la forme *a->e[n]*. Les pointeurs d'événement et les pointeurs de liste sont ainsi les outils de base pour manipuler les données d'un fichier de partition. Les pointeurs et les listes de pointeurs peuvent être copiés et réordonnés sans modifier les valeurs des données auxquelles ils font référence. Cela signifie que l'on peut copier et manipuler les notes et les phrases depuis un niveau de contrôle élevé. Alternativement, les données d'un événement ou d'un groupe d'événements peuvent être modifiées sans changer les pointeurs d'événement ou de liste. Les fonctions de l'API *Cscore* permettent de créer et de manipuler des partitions de cette manière.

Avec Csound 5, les noms de toutes les fonctions de l'API *Cscore* ont été changés pour être plus explicites. De plus, chaque fonction nécessite maintenant un pointeur sur un objet CSOUND en premier argument. La structure de l'objet CSOUND n'a pas d'importance (en fait il ne peut pas être modifié dans un programme utilisateur). Le moyen d'obtenir ce pointeur sur un objet CSOUND sera montré dans la section suivante. Les fonctions de *Cscore* et ses structures de données sont déclarées dans le fichier d'en-tête `cscore.h` que vous devez inclure dans le code de votre programme avant leur utilisation.

Les noms des fonctions de *Cscore* spécifient si elles opèrent sur des événements ou sur des listes d'événements. Dans le sommaire suivant des appels de fonction disponibles, on utilise quelques conventions de nommage :

Le symbole cs est un pointeur vers un objet CSOUND (CSOUND \*);  
Les symboles e, f sont des pointeurs sur des événements (notes);  
Les symboles a, b sont des pointeurs sur des listes (arrays) de tels événements;  
Le symbole n est un paramètre entier de type int;  
"..." indique un paramètre chaîne (soit une constante soit une variable de type char \*);  
Le symbole fp est un pointeur sur un fichier (FILE \*) en flot d'entrée de partition;

syntaxe d'appel	description
-----	-----
/* Fonctions pour travailler avec des événements */	
e = cscoreCreateEvent(cs, n);	crée un événement vide avec n pchamps
e = cscoreDefineEvent(cs, "...");	définit un événement par la chaîne de caractères ...
e = cscoreCopyEvent(cs, f);	fait une nouvelle copie de l'événement f
e = cscoreGetEvent(cs);	lit l'événement suivant dans le fichier de partition en
cscorePutEvent(cs, e);	écrit l'événement e dans le fichier de partition en sor
cscorePutString(cs, "...");	écrit l'événement défini par la chaîne dans la partiti
	en sortie
/* Fonctions pour travailler avec des listes d'événements */	
a = cscoreListCreate(cs, n);	crée une liste d'événements vide avec n emplacements
a = cscoreListAppendEvent(cs, a, e);	ajoute l'événement e à la fin de la liste a
a = cscoreListAppendStringEvent(cs, a, "...");	ajoute l'événement défini par la chaîne à la liste a
a = cscoreListCopy(cs, b);	copie la liste b (mais pas les événements)
a = cscoreListCopyEvents(cs, b);	copie les événements de b, en créant une nouvelle liste
a = cscoreListGetSection(cs);	lit tous les événements de la partition en entrée, jusqu
	prochain s ou e
a = cscoreListGetNext(cs, nbeats);	lit les prochaines nbeats pulsations de la partition en
	(nbeats est un MYFLT)
a = cscoreListGetUntil(cs, beatno);	lit tous les événements de la partition en entrée jusqu
	pulsation beatno (MYFLT)
a = cscoreListSeparateF(cs, b);	sépare les instructions f de la liste b vers la liste a
a = cscoreListSeparateTWF(cs, b);	sépare les instructions t,w & f de la liste b vers la l
a = cscoreListAppendList(cs, a, b);	ajoute la liste b à la liste a
a = cscoreListConcatenate(cs, a, b);	concaténation des listes a et b (identique au précédent
cscoreListSort(cs, a);	trie la liste a en ordre chronologique selon p[2]
n = cscoreListCount(cs, a);	retourne le nombre d'événements dans la liste a
a = cscoreListExtractInstruments(cs, b, "...");	extrait les notes des instruments ... (pas de nouveaux
	événements)
a = cscoreListExtractTime(cs, b, from, to);	extrait les notes d'une période de temps, en créant de
	nouveaux événements (from et to sont des MYFLT)
cscoreListPut(cs, a);	écrit les événements de la liste a dans le fichier de p
	sortie
cscoreListPlay(cs, a);	envoie les événements de la liste a vers l'orchestre de
	une exécution immédiate (ou les imprime s'il n'y a pas
/* Fonctions pour réclamer de la mémoire */	
cscoreFreeEvent(cs, e);	libère l'espace de l'événement e
cscoreListFree(cs, a);	libère l'espace de la liste a (mais pas les événements)
cscoreListFreeEvents(cs, a);	libère les événements de la liste a, et l'espace de la
/* Fonctions pour travailler avec plusieurs fichiers de partition en entrée */	
fp = cscoreFileGetCurrent(cs);	recupère le pointeur du fichier de partition en entrée
	actif (au départ trouve le pointeur du fichier de par
	entrée de la ligne de commande)
fp = cscoreFileOpen(cs, "filename");	ouvre un autre fichier de partition en entrée (5 au max
cscoreFileSetCurrent(cs, fp);	fait de fp le pointeur sur le fichier de partition
	actuellement actif
cscoreFileClose(cs, fp);	ferme le fichier de partition en relation avec FILE *fp

Sous Csound 4, les noms des fonctions et leurs paramètres étaient les suivants :

syntaxe d'appel	description
-----	-----
e = createv(n);	crée un événement vide avec n pchamps
e = defev("...");	définit un événement par la chaîne de caractères ...
e = copyev(f);	fait une nouvelle copie de l'événement f
e = getev();	lit l'événement suivant dans le fichier de partition en entrée
putev(e);	écrit l'événement e dans le fichier de partition en sortie
putstr("...");	écrit l'événement défini par la chaîne dans la partition en sortie
a = lcreat(n);	crée une liste d'événements vide avec n emplacements

<pre>int n; a = lappev(a,e); a = lappstrev(a,"..."); a = lcopy(b); a = lcopyev(b); a = lget();  a = lgetnext(nbeats); float nbeats; a = lgetuntil(beatno);  float beatno; a = lsepf(b); a = lseptwf(b); a = lcat(a,b); lsort(a); a = lxins(b,"..."); a = lxtimev(b,from,to); float from, to; lput(a); lplay(a);  relev(e); lrel(a); lrelev(a); fp = getcurfp();  fp = filopen("filename"); setcurfp(fp); filclose(fp);</pre>	<p>ajoute l'évènement e à la fin de la liste a</p> <p>ajoute l'évènement défini par la chaîne à la liste a</p> <p>copie la liste b (mais pas les évènements)</p> <p>copie les évènements de b, en créant une nouvelle liste</p> <p>lit tous les évènements de la partition en entrée, jusqu'au prochain s ou e</p> <p>lit les prochaines nbeats pulsations de la partition en entrée</p> <p>lit tous les évènements de la partition en entrée jusqu'à la pulsation beatno</p> <p>sépare les instructions f de la liste b vers la liste a</p> <p>sépare les instructions t,w &amp; f de la liste b vers la liste a</p> <p>concaténation (ajout) de la liste b à la liste a</p> <p>trie la liste a en ordre chronologique selon p[2]</p> <p>extraît les notes des instruments ... (pas de nouveaux évènements)</p> <p>extraît les notes d'une période de temps, en créant de nouveaux évènements</p> <p>écrit les évènements de la liste a dans le fichier de partition en sortie</p> <p>envoie les évènements de la liste a vers l'orchestre de Csound pour une exécution immédiate (ou les imprime s'il n'y a pas d'orchestre)</p> <p>libère l'espace de l'évènement e</p> <p>libère l'espace de la liste a (mais pas les évènements)</p> <p>libère les évènements de la liste a, et l'espace de la liste</p> <p>récupère le pointeur du fichier de partition en entrée actuellement actif (au départ trouve le pointeur du fichier de partition en entrée de la ligne de commande)</p> <p>ouvre un autre fichier de partition en entrée (5 au maximum)</p> <p>fait de fp le pointeur sur le fichier de partition actuellement actif</p> <p>ferme le fichier de partition en relation avec FILE *fp</p>
--	---

## Ecrire un Programme de Contrôle Cscore

Le format général d'un programme de contrôle *Cscore* est :

```
#include "cscore.h"  
void cscore(CSOUND *cs)  
{  
    /* DECLARATIONS DES VARIABLES */  
    /* CORPS DU PROGRAMME */  
}
```

L'instruction *include* définira les structures d'évènement et de liste et toutes les fonctions de l'API *Cscore* pour le programme. Il faut que le nom de la fonction de l'utilisateur soit *cscore* si elle doit être liée avec le programme *main* standard dans *cscormai.c* ou liée comme routine *Cscore* interne pour un exécutable de Csound personnalisé. Cette fonction *cscore()* reçoit un argument de *cscormai.c* ou de Csound -- *CSOUND \*cs* -- qui est un pointeur sur un objet Csound. Le pointeur *cs* doit être passé en premier paramètre à toutes les fonctions de l'API *Cscore* que le programme appelle.

Le programme C suivant lira depuis une *partition numérique standard*, jusqu'à (mais sans l'inclure) la première *instruction s* ou *e*, puis il écrira ces données (inchangées) en sortie.

```
#include "cscore.h"  
void cscore(CSOUND *cs)  
{  
    EVLIST *a; /* a est autorisé à pointer sur une liste d'évènements */  
    a = cscoreListGetSection(cs); /* lit les évènements, retourne le pointeur de liste */  
    cscoreListPut(cs, a); /* écrit ces évènements en sortie (inchangés) */  
    cscorePutString(cs, "e"); /* écrit la chaîne e sur la sortie */  
}
```



Après l'exécution de `cscoreListGetSection()`, la variable `a` pointe sur une liste d'adresses d'évènements, qui pointent chacune sur un évènement stocké. Nous avons utilisé ce même pointeur pour permettre à une autre fonction de liste -- `cscoreListPut()` -- d'accéder à tous les évènements qui ont été lus et de les écrire en sortie. Si nous définissons maintenant un autre symbole `e` comme pointeur d'évènement, alors l'instruction

```
e = a->e[4];
```

lui affectera le contenu du 4ème emplacement de la structure `EVLIST`, `a`. Ce contenu est un pointeur sur un évènement, qui comprend lui-même un tableau de valeurs de champs de paramètre. Ainsi le terme `e->p[5]` signifiera la valeur du champ de paramètre 5 du 4ème évènement dans la `EVLIST` dénotée par `a`. Le programme ci-dessous multipliera la valeur de ce *p-champ* par 2 avant de l'écrire en sortie.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
    EVENT *e;                                /* un pointeur sur un évènement */
    EVLIST *a;
    a = cscoreListGetSection(cs); /* lit une partition comme une liste d'évènements */
    e = a->e[4];                        /* pointe sur l'évènement 4 dans la liste a */
    e->p[5] *= 2;                        /* trouve le p-champ 5, multiplie sa valeur par 2 */
    cscoreListPut(cs, a);               /* écrit en sortie la liste d'évènements */
    cscorePutString(cs, "e");           /* ajoute une instruction de "fin de partition" */
}
```

Considérez maintenant la partition suivante, dans laquelle `p[5]` contient la fréquence en Hz.

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
e
```

Si cette partition est donnée au programme principal précédent, la sortie résultante ressemblera à ceci :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
i 1 1 3 0 440 10000
i 1 4 3 0 512 10000      ; p[5] est devenu 512 au lieu de 256.
i 1 7 3 0 880 10000
e
```

Notez que le 4ème évènement est en fait la seconde note de la partition. Jusqu'ici nous n'avons pas fait de distinction entre les notes et les tables de fonction mises en place dans une partition numérique. Les deux peuvent être classées comme évènement. Notez aussi que notre 4ème évènement a été stocké dans le champ `e[4]` de la structure. Pour être compatible avec la notation des *p-champs* de Csound, nous ignorerons `p[0]` et `e[0]` dans les structures d'évènement et de liste, en stockant `p1` dans `p[1]`, l'évènement 1 dans `e[1]`, etc. Les fonctions de *Cscore* adoptent toutes cette convention.

Pour étendre l'exemple ci-dessus, nous pourrions décider d'utiliser les mêmes pointeurs `a` et `e` pour examiner chacun des évènements dans la liste. Noter que `e` n'a pas été fixé au nombre 4, mais au contenu du 4ème emplacement de la liste. Pour inspecter le `p5` de l'évènement précédent dans la liste, nous n'avons

qu'à redéfinir *e* avec l'affectation

```
e = a->e[3];
```

et référencer le 5ème emplacement du tableau de *p-champs* avec l'expression

```
e->p[5]
```

Plus généralement, nous pouvons utiliser une variable entière comme indice du tableau *e[]*, et accéder séquentiellement à chaque évènement en utilisant une boucle et en incrémentant l'indice. Le nombre d'évènements stockés dans une *EVLIST* est contenu dans le membre *nevents* de la structure.

```
int index; /* démarre avec e[1] car e[0] n'est pas utilisé */
for (index = 1; index <= a->nevents; index++)
{
    e = a->e[index];
    /* faire quelque chose avec e */
}
```

L'exemple ci-dessus démarre avec *e[1]* et augmente l'indice à chaque passage dans la boucle (*index++*) jusqu'à ce qu'il soit plus grand que *a->nevents*, l'indice du dernier évènement dans la liste. Les instructions à l'intérieur de la boucle *for* sont exécutées une dernière fois quand *index* égale *a->nevents*.

Dans le programme suivant nous utiliserons la même partition en entrée. Cette fois nous séparerons les instructions de *ftable* des instructions de *note*. Nous écrirons ensuite en sortie les trois évènements de *note* stockés dans la liste *a*, puis nous créerons une seconde section de partition constituée de l'ensemble de hauteurs original et d'une version transposée de celui-ci. Cela apportera un doublement à l'octave.

Ici, notre indice dans le tableau est *n* et il est incrémenté dans un bloc *for* qui boucle *nevents* fois, ce qui permet d'appliquer une instruction au même *p-champ* des évènements successifs.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
    EVENT *e, *f;
    EVLIST *a, *b;
    int n;

    a = cscoreListGetSection(cs);
    b = cscoreListSeparateF(cs, a);
    cscoreListPut(cs, b);
    e = cscoreDefineEvent(cs, "t 0 120");
    cscorePutEvent(cs, e);
    cscoreListPut(cs, a);
    cscorePutString(cs, "s");
    cscorePutEvent(cs, e);
    b = cscoreListCopyEvents(cs, a);
    for (n = 1; n <= b->nevents; n++)
    {
        f = b->e[n];
        f->p[5] *= 0.5;
    }
    a = cscoreListAppendList(cs, a, b);
    cscoreListPut(cs, a);
    cscorePutString(cs, "e");
}
```

```
/* lit la partition dans la liste d'évènements "a" */
/* sépare les instructions f */
/* écrit les instructions f dans la partition en sortie */
/* définit un évènement pour l'instruction de tempo */
/* écrit l'instruction de tempo dans la partition */
/* écrit les notes */
/* fin de section */
/* écrit l'instruction de tempo encore une fois */
/* fait une copie des notes dans "a" */
/* répète les lignes suivantes nevents fois : */

/* transpose la hauteur d'une octave vers le bas */

/* ajoute ces notes aux hauteurs originales */
```

La sortie de ce programme est :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
s
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
i 1 1 3 0 220 10000
i 1 4 3 0 128 10000
i 1 7 3 0 440 10000
e
```

Si la sortie est écrite dans un fichier, le fait que les événements ne soient pas ordonnés n'est pas un problème. La sortie est écrite dans un fichier (ou sur la sortie standard) chaque fois que la fonction *cscoreListPut()* est utilisée. Cependant, si ce programme était appelé durant une exécution de Csound et que la fonction *cscoreListPlay()* était remplacée par *cscoreListPut()*, alors les événements seraient envoyés à l'orchestre au lieu du fichier et il faudrait qu'ils soient préalablement triés en appelant la fonction *cscoreListSort()*. Les détails de la sortie de la partition et de son exécution quand on utilise *Cscore* depuis Csound sont décrits dans la section suivante.

Ensuite nous étendons le programme ci-dessus en utilisant la boucle *for* pour lire *p[5]* et *p[6]*. Dans la partition originale *p[6]* dénote l'amplitude. Pour créer un diminuendo sur l'octave inférieure ajoutée, qui soit indépendant de l'ensemble de notes original, une variable appelée *dim* sera utilisée.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
    EVENT *e, *f;
    EVLIST *a, *b;
    int n, dim;                                /* déclare deux variables entières */

    a = cscoreListGetSection(cs);
    b = cscoreListSeparateF(cs, a);
    cscoreListPut(cs, b);
    cscoreListFreeEvents(cs, b);
    e = cscoreDefineEvent(cs, "t 0 120");
    cscorePutEvent(cs, e);
    cscoreListPut(cs, a);
    cscorePutString(cs, "s");
    cscorePutEvent(cs, e);                    /* écrit une autre instruction de tempo */
    b = cscoreListCopyEvents(cs, a);
    dim = 0;                                /* initialise dim à 0 */
    for (n = 1; n <= b->nevents; n++)
    {
        f = b->e[n];
        f->p[6] -= dim;                      /* soustrait la valeur courante de dim */
        f->p[5] *= 0.5;                      /* transpose la hauteur une octave plus bas */
        dim += 2000;                        /* augmente dim pour chaque note */
    }
    a = cscoreListAppendList(cs, a, b);    /* ajoute ces notes aux hauteurs originales */
    cscoreListPut(cs, a);
    cscorePutString(cs, "e");
}
```

En utilisant à nouveau la même partition en entrée, la sortie de ce programme est :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
s
t 0 120
i 1 1 3 0 440 10000      ; Trois notes originales aux pulsations
i 1 4 3 0 256 10000      ; 1, 4 et 7 sans diminuendo.
i 1 7 3 0 880 10000
i 1 1 3 0 220 10000      ; Trois notes transposées une octave plus bas
i 1 4 3 0 128 8000       ; également aux pulsations 1, 4 et 7
i 1 7 3 0 440 6000       ; avec diminuendo.
e
```

Dans le programme suivant la même séquence de trois notes sera répétée à divers intervalles de temps. La date de début de chaque groupe est déterminée par les valeurs du tableau *cue*. Cette fois le *dim* se produira sur chaque groupe de notes plutôt que sur chaque note. Remarquez la position de l'instruction qui incrémente la variable *dim* en dehors de la boucle *for* intérieure.

```
#include "cscore.h"
int cue[3] = {0,10,17};          /* déclare un tableau de 3 entiers */
void cscore(CSOUND *cs)
{
    EVENT *e, *f;
    EVLIST *a, *b;
    int n, dim, cuecount;        /* déclare la nouvelle variable cuecount */

    a = cscoreListGetSection(cs);
    b = cscoreListSeparateF(cs, a);
    cscoreListPut(cs, b);
    cscoreListFreeEvents(cs, b);
    e = cscoreDefineEvent(cs, "t 0 120");
    cscorePutEvent(cs, e);
    dim = 0;
    for (cuecount = 0; cuecount <= 2; cuecount++) /* les éléments de cue sont numérotés 0, 1, 2 */
    {
        for (n = 1; n <= a->nevents; n++)
        {
            f = a->e[n];
            f->p[6] -= dim;
            f->p[2] += cue[cuecount];          /* ajoute les valeurs de cue */
        }
        printf("; diagnostic: cue = %d\n", cue[cuecount]);
        dim += 2000;
        cscoreListPut(cs, a);
    }
    cscorePutString(cs, "e");
}
```

Ici la boucle *for* intérieure lit les événements de la liste *a* (les notes) et la boucle *for* extérieure lit chaque répétition des événements de la liste *a* (les "répliques" du groupe de hauteurs). Ce programme démontre aussi un moyen utile de résolution de problème au moyen de la fonction *printf*. Le *point-virgule* commence la chaîne de caractères pour produire un commentaire dans le fichier de partition résultant. Dans ce cas, la valeur de *cue* est imprimée en sortie pour s'assurer que le programme prend le bon membre du tableau au bon moment. Lorsque les données de sortie sont fausses ou que des messages d'erreur sont rencontrés, la fonction *printf* peut aider à identifier le problème.

A partir du même fichier d'entrée, le programme C ci-dessus générera la partition suivante. Pouvez-vous expliquer pourquoi le dernier ensemble de notes ne démarre pas au bon moment et comment corriger le problème ?

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
; diagnostic: cue = 0
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
; diagnostic: cue = 10
i 1 11 3 0 440 8000
i 1 14 3 0 256 8000
i 1 17 3 0 880 8000
; diagnostic: cue = 17
i 1 28 3 0 440 4000
i 1 31 3 0 256 4000
i 1 34 3 0 880 4000
e
```

## Compiler un Programme Cscore

Un programme *Cscore* peut être invoqué comme un *programme autonome* ou comme une partie de Csound placée entre le tri de la partition et son exécution par l'orchestre :

```
cscore [fichier_partition_entrée] [> fichier_partition_sortie]
```

ou

```
csound [-C] [autresoptions] [nomorch] [nompartition]
```

Avant d'essayer de compiler votre propre programme *Cscore*, vous voudrez sans doute obtenir une copie du code source de Csound. Téléchargez la distribution des sources la plus récente pour votre plate-forme ou bien récupérez (check out) une copie du module csound5 depuis le CVS de Sourceforge. Il y a plusieurs fichiers dans les sources qui vous aideront. Il y a dans le répertoire *examples/cscore/* plusieurs exemples de programmes de contrôle *Cscore*, y compris tous les exemples contenus dans ce manuel. Et il y a dans le répertoire *frontends/cscore/* les deux fichiers *cscoremain.c* et *cscore.c*. *cscoremain.c* contient une simple fonction *main* qui réalise toute l'initialisation qu'un programme *Cscore* autonome doit faire avant d'appeler votre fonction de contrôle. Cette « souche » *main* initialise Csound, lit les arguments de la ligne de commande, ouvre les fichiers de partition en entrée et en sortie, et appelle ensuite une fonction *cscore()*. Comme il est décrit ci-dessus, vous êtes chargé d'écrire la fonction *cscore()* et de la fournir dans un autre fichier. Le fichier *frontends/cscore/cscore.c* montre l'exemple le plus simple d'une fonction *cscore()* qui lit une partition de n'importe quelle longueur et l'écrit inchangée sur la sortie.

Ainsi, pour créer un programme autonome, écrivez un programme de contrôle en suivant les indications de la section précédente. Supposons que vous ayez sauvegardé ce programme dans un fichier nommé "*mycscore.c*". Vous devez ensuite compiler ce programme et le lier avec la bibliothèque de Csound et *cscoremain.c* pour créer un exécutable, en suivant l'ensemble de directives ci-dessous qui s'applique à votre système d'exploitation. Il sera utile d'avoir une certaine familiarité avec le compilateur C de votre ordinateur car l'information ci-dessous ne peut pas être exhaustive pour tous les systèmes existants.

## Linux et Unix

Les commandes suivantes supposent que vous ayez copié votre fichier *mycscore.c* dans le même répertoire que *cscoremain.c*, que vous ayez ouvert un terminal sur ce même répertoire et que vous ayez installé au préalable une distribution binaire de Csound qui aura placé une bibliothèque *libcsound.a* ou *libcsound.so* dans */usr/local/lib* et les fichiers d'en-tête pour l'API de Csound dans */usr/local/include/csound*.

Pour la compilation et l'édition de liens, tapez :

```
gcc mycscore.c cscoremain.c -o cscore -lcsound -L/usr/local/lib -I/usr/local/include/csound
```

Pour l'exécution (avec envoi des résultats sur la sortie standard), tapez :

```
./cscore test.sco
```

Il est possible que sur certains systèmes Unix le compilateur C soit nommé *cc* ou quelque chose d'autre que *gcc*.

## Windows

Csound est ordinairement compilé sur Windows au moyen de l'environnement MinGW qui fournit GCC -- le même compilateur utilisé sur Linux -- au travers d'un shell de commande (MSYS) à la Unix. Comme les bibliothèques pré-compilées pour Csound sur Windows sont construites de cette manière, vous utiliserez probablement MinGW pour la liaison avec celles-ci. Si vous avez construit Csound en utilisant un autre compilateur, vous serez sans doute capable de construire également *Cscore* avec ce compilateur.

La compilation de programmes *Cscore* autonomes en utilisant MinGW devrait être similaire à la procédure ci-dessus pour Linux avec les chemins de la bibliothèque et des en-têtes changés pour pointer là où Csound est installé sur le système Windows. *(Les contributions plus détaillées sur ces instructions seront les bienvenues car le rédacteur de cet article n'a pas pu tester Cscore sur une machine Windows).*

## OS X

Les commandes suivantes supposent que vous ayez copié votre fichier *mycscore.c* dans le même répertoire que *cscoremain.c* et que vous ayez ouvert un terminal dans ce répertoire. De plus, les outils de développement fournis par Apple (incluant le compilateur GCC) doivent être installés sur votre système et vous devez avoir installé une distribution binaire de Csound qui aura placé le framework Csoundlib dans */Library/Frameworks*.

Utilisez cette commande pour la compilation et l'édition de liens. (Il peut y avoir un avertissement sur de "multiples définitions du symbole \_cscore").

```
gcc cscore.c cscoremain.c -o cscore -framework CsoundLib -I/Library/Frameworks/CsoundLib.framework/Head
```

Pour l'exécution (avec envoi des résultats sur la sortie standard) :

```
./cscore test.sco
```

## MacOS 9

Vous devrez avoir installé CodeWarrior ou un autre environnement de développement sur votre ordinateur (MPW peut fonctionner). Téléchargez la distribution des sources pour OS 9 (elle aura un nom comme *Csound5.05\_OS9\_src.smi.bin*).

Si vous utiliser CodeWarrior, trouvez et ouvrez le fichier de projet "Cscore5.cw8.mcp" dans le répertoire "Csound5.04-OS9-source;macintosh:Csound5Library:". Ce fichier de projet est configuré pour utiliser les fichiers source *cscore.c* et *cscoremain\_MacOS9.c* situés dans l'arborescence des sources csound5 et la librairie partagée Csound5Lib produite lors de la compilation de Csound avec le fichier de projet "Csound5.cw8.mcp". Il vous faut substituer votre propre fichier du programme *Cscore* à la place de *cscore.c* et soit avoir compilé Csound5Lib avant, soit substituer une copie de la bibliothèque dans le projet à partir de la distribution binaire de Csound pour OS 9. Le fichier *cscoremain\_MacOS9.c* contient du code spécialisé pour la configuration de la bibliothèque de console SIOUX de CodeWarrior et permet l'entrée d'arguments de ligne de commande avant le lancement du programme.

Une fois que les fichiers nécessaires sont inclus dans la fenêtre du projet, cliquez sur le bouton "Make" et CodeWarrior produira une application nommée « *Cscore* ». Quand vous lancez cette application, elle affiche d'abord une fenêtre vous permettant de saisir les arguments pour la fonction principale. Vous n'avez qu'à taper le nom de fichier ou le nom de chemin complet de la partition en entrée -- ne tapez pas "cscore". Le fichier d'entrée doit se trouver dans le même répertoire que l'application sinon vous devrez taper un chemin complet ou relatif pour le fichier. La sortie sera affichée dans la fenêtre de console. Vous pouvez utiliser la commande *Save* du menu *File* avant de quitter la console. Alternativement, dans la fenêtre de dialogue de la ligne de commande, vous pouvez choisir de rediriger la sortie dans un fichier en cliquant sur le bouton *File* sur le côté droit de la fenêtre de dialogue. (Notez que la fenêtre de console ne peut afficher qu'environ 32000 caractères, ce qui rend l'écriture dans un fichier nécessaire pour les grandes partitions).

## Rendre Cscore utilisable depuis Csound

Pour opérer depuis Csound, suivez d'abord les instructions pour compiler Csound (voir *Construire Csound*) qui concernent le système d'exploitation que vous utilisez. Une fois que vous avez réussi à construire un système Csound non modifié, substituez alors votre propre fonction *cscore()* à celle qui se trouve dans le fichier *Top/cscore\_internal.c*, et reconstruisez Csound.

L'exécutable résultant est votre Csound spécial, utilisable comme ci-dessus. L'option *-C* invoquera votre programme *Cscore* après le tri de la partition d'entrée dans « *score.srt* ». Les détails de ce qui se passe lorsque vous lancez Csound avec l'option *-C* flag sont donnés dans la section suivante.

Csound 5 fournit aussi un moyen supplémentaire d'exécuter votre propre programme *Cscore* depuis Csound. En utilisant l'API, une application hôte peut mettre en place une *fonction d'appel en retour* (*callback*) de *Cscore*, qui est une fonction que Csound appellera à la place de sa fonction interne *cscore()*. L'avantage de cette approche est qu'il n'est pas nécessaire de recompiler la totalité de Csound. Un autre bénéfice est que l'application hôte peut choisir pendant l'exécution la fonction de callback parmi plusieurs fonctions *Cscore*. L'inconvénient est que vous devez écrire une application hôte.

Une approche simple pour utiliser un callback *Cscore* via l'API serait de modifier le programme main standard de Csound -- qui est un hôte simple de Csound -- contenu dans le fichier *frontends/csound/csound\_main.c*. L'ajout d'un appel à *csoundSetCscoreCallback()* après l'appel à *csoundCreate()* mais avant l'appel à *csoundCompile()* devrait faire l'affaire. En recompilant ce fichier et en le liant à une bibliothèque de Csound existante, on obtiendra une version de Csound en ligne de commande qui fonctionne comme celle qui est décrite ci-dessus. N'oubliez pas de taper l'option *-C*.

## Notes au sujet des formats de partition et du comportement de l'exécutable

Comme indiqué précédemment, les fichiers d'entrée de *Cscore* peuvent se trouver dans leur forme originale ou résolue en temps et pré-triée ; cette modalité sera préservée (section par section) lors de la lecture, du traitement et de l'écriture des partitions. Le traitement autonome utilisera le plus souvent des sources non résolues en temps et créera de nouveau fichiers de même forme. Lors du traitement depuis Csound, la partition en entrée arrivera déjà résolue en temps et triée, et pourra ainsi être envoyée directement (normalement section par section) à l'orchestre. Un des avantages de cette façon d'utiliser *Cscore*

est que toutes les commodités de syntaxe du langage de partition complet de Csound peuvent être utilisées -- macros, expressions arithmétiques, carry, rampes, etc. -- car la partition passera par les phases "Carry, Tempo, Tri" du traitement avant d'être transmise au programme *Cscore* fourni par l'utilisateur.

Lors du traitement dans Csound, une liste d'événements peut être transmise à un orchestre de Csound en utilisant *cscoreListPlay()*. Il peut y avoir n'importe quel nombre d'appels de *cscoreListPlay()* dans un programme *Cscore*. Chaque liste ainsi transmise peut-être résolue ou non en temps, mais chaque liste doit être en ordre chronologique strict par rapport à *p2* (soit grâce au pré-traitement de tri soit en utilisant *cscoreListSort()*). S'il n'y a pas de *cscoreListPlay()* dans un module *Cscore* exécuté depuis Csound, tous les événements écrits en sortie (via *cscorePutEvent()*, *cscorePutString()*, ou *cscoreListPut()*) sont envoyés dans une nouvelle partition dans le répertoire courant nommée « *cscore.out* ». Csound invoque alors à nouveau le tri de partition avant d'envoyer cette nouvelle partition à l'orchestre pour son exécution. La partition de sortie triée finale est écrite dans un fichier nommé « *cscore.srt* ».

Un programme *Cscore* autonome utilisera normalement la commande « put » pour écrire dans son fichier de sortie. Si un programme *Cscore* autonome appelle *cscoreListPlay()*, les événements ainsi destinés à l'exécution seront envoyés sur la sortie comme s'ils provenaient de *cscoreListPut()*.

Une liste de notes envoyée par *cscoreListPlay()* pour exécution doit être distincte dans le temps des listes de notes suivantes. Aucune fin de note ne doit dépasser la date de début de la liste suivante, car *cscoreListPlay()* complètera chaque liste avant d'attaquer la suivante (comme un marqueur de Section qui ne réinitialise pas le temps local à zéro). C'est important lorsque l'on utilise *cscoreListGetNext()* ou *cscoreListGetUntil()* pour charger et traiter des segments de partition avant exécution, car ces fonctions pourraient ne lire qu'une partie d'une section non triée.

## Exemples Plus Avancés

Le programme suivant démontre la lecture à partir de deux fichiers d'entrée différents. L'idée est d'alterner entre deux partitions de 2 sections, et d'écrire les sections entrelacées dans un seul fichier de sortie.

```
#include "cscore.h"                /* CSORE_SWITCH.C */
cscore(CSOUND* cs)                /* appellable depuis Csound ou comme cscore autonome */
{
    EVLIST *a, *b;
    FILE *fp1, *fp2;
    fp1 = cscoreFileGetCurrent(cs); /* deux pointeurs sur des flots de fichier de partition */
    fp2 = cscoreFileOpen(cs, "score2.srt"); /* la partition de la ligne de commande */
    a = cscoreListGetSection(cs); /* une partition supplémentaire */
    cscoreListPut(cs, a); /* lit une section de la partition 1 */
    cscorePutString(cs, "s"); /* l'écrit en sortie telle quelle */
    cscoreFileSetCurrent(cs, fp2);
    b = cscoreListGetSection(cs); /* lit une section de la partition 2 */
    cscoreListPut(cs, b); /* l'écrit en sortie telle quelle */
    cscorePutString(cs, "s");
    cscoreListFreeEvents(cs, a); /* facultatif, pour libérer de l'espace */
    cscoreListFreeEvents(cs, b);
    cscoreFileSetCurrent(cs, fp1);
    a = cscoreListGetSection(cs); /* lit la section suivante de la partition 1 */
    cscoreListPut(cs, a); /* l'écrit en sortie */
    cscorePutString(cs, "s");
    cscoreFileSetCurrent(cs, fp2);
    b = cscoreListGetSection(cs); /* lit la section suivante de la partition 2 */
    cscoreListPut(cs, b); /* l'écrit en sortie */
    cscorePutString(cs, "e");
}
```

Finalement, nous montrons comment prendre un fichier de partition littérale, non interprétée et lui insuffler un peu d'expressivité rythmique. La théorie des pulsations métriques liées au compositeur a été étudiée en profondeur par Manfred Clynes, et la suite est dans l'esprit de ce travail. Ici, la stratégie consiste à créer d'abord un *tableau* de nouvelles dates de *début* pour chaque début possible de double croche,



puis par indexation dans ce tableau, d'ajuster le début et la durée de chaque note de la partition d'entrée aux dates interprétées. On montre aussi comment un orchestre de Csound peut être invoqué de façon répétitive depuis un générateur de partition pendant l'exécution.

```
#include "cscore.h"                /*  CSCORE_PULSE.C  */

/* programme pour appliquer une pulsation aux durées interprétées */
/* à une partition existante en 3/4, premiers temps sur 0, 3, 6 ... */

static float four[4] = { 1.05, 0.97, 1.03, 0.95 }; /* largeur de pulsation des 4 */
static float three[3] = { 1.03, 1.05, .92 };      /* largeur de pulsation des 3 */

cscore(CSOUND* cs)                  /* Cet exemple doit être appelé depuis Csound */
{
    EVLIST *a, *b;
    EVENT *e, **ep;
    float pulse16[4*4*4*4*3*4]; /* tableau de doubles croches, 3/4, 256 mesures */
    float acc16, acc1, inc1, acc3, inc3, acc12, inc12, acc48, inc48, acc192, inc192;
    float *p = pulse16;
    int n16, n1, n3, n12, n48, n192;

    /* remplit le tableau avec les dates de début de l'interprétation */
    for (acc192=0., n192=0; n192<4; acc192+=192.*inc192, n192++)
        for (acc48=acc192, inc192=four[n192], n48=0; n48<4; acc48+=48.*inc48, n48++)
            for (acc12=acc48, inc48=inc192*four[n48], n12=0; n12<4; acc12+=12.*inc12, n12++)
                for (acc3=acc12, inc12=inc48*four[n12], n3=0; n3<4; acc3+=3.*inc3, n3++)
                    for (acc1=acc3, inc3=inc12*four[n3], n1=0; n1<3; acc1+=inc1, n1++)
                        for (acc16=acc1, inc1=inc3*three[n1], n16=0; n16<4; acc16+=.25*inc1*four[n16], n16++)
                            *p++ = acc16;

    /* for (p = pulse16, n1 = 48; n1--; p += 4) /* montre les valeurs & les différences */
    /* printf("%g %g %g %g %g %g %g %g %g\n", *p, *(p+1), *(p+2), *(p+3), */
    /* *(p+1)-*p, *(p+2)-*(p+1), *(p+3)-*(p+2), *(p+4)-*(p+3)); */

    a = cscoreListGetSection(cs); /* lit une section de la partition résolue en temps */
    b = cscoreListSeparateTWF(cs, a); /* sépare les instructions de jeu et de fonction */
    cscoreListPlay(cs, b); /* et les envoie à l'exécution */
    a = cscoreListAppendStringEvent(cs, a, "s"); /* ajoute une instruction de section à la liste de notes */
    cscoreListPlay(cs, a); /* joue la liste de notes sans interprétation */
    for (ep = &a->e[1], n1 = a->nevents; n1--; ) { /* maintenant modifie les pulsations */
        e = *ep++;
        if (e->op == 'i') {
            e->p[2] = pulse16[(int)(4. * e->p2orig)];
            e->p[3] = pulse16[(int)(4. * (e->p2orig + e->p3orig))] - e->p[2];
        }
    }

    cscoreListPlay(cs, a); /* maintenant joue la liste modifiée */
}
```

---

# Etendre Csound

## Ajouter des Générateurs Unitaires

Si les générateurs unitaires existants de Csound ne répondent pas à vos besoins, il est relativement aisé d'étendre Csound en écrivant de nouveaux générateurs unitaires en C ou en C++. Le traducteur, le chargeur et le moniteur d'exécution traiteront votre module comme n'importe quel autre module, pourvu que vous suiviez certaines conventions.

Historiquement, on réalisait ceci avec des générateurs unitaires intégrés, c'est-à-dire dont le code est lié statiquement avec le reste de l'exécutable de Csound.

Aujourd'hui, on préfère créer des générateurs unitaires sous forme de plugin. Ce sont des bibliothèques à liaison dynamique (DLL) sous Windows, et des modules chargeables (bibliothèques partagées chargées par `dlopen`) sur Linux. Csound recherche et charge ces plugins au moment de l'exécution. L'avantage de cette méthode, naturellement, est que les plugins créés par n'importe quel développeur, n'importe quand, peuvent être utilisés avec des versions de Csound déjà existantes.

## Créer un Générateur Unitaire Intégré

Vous avez besoin d'une structure définissant les entrées, les sorties et l'espace de travail, plus du code d'initialisation et du code d'exécution. Mettons un exemple de tout cela dans deux nouveaux fichiers, `newgen.h` et `newgen.c`. Les exemples donnés sont pour Csound 5. Pour les versions antérieures, il faut omettre le premier paramètre (`CSOUND *csound`) dans toutes les fonctions d'opcode.

```
/* newgen.h - définit une structure */

/* Déclare les structures et les fonctions de Csound. */
#include "csoundCore.h"

typedef struct
{
    OPDS h;
    MYFLT *result, *istrt, *incr, *itime, *icontin; /* en-tête requis */
    MYFLT curval, vincr; /* adr des arg de sortie et d'entrée */
    long countdown; /* espace de données privé */
} RMP; /* ditto */

/* newgen.c - code d'initialisation et d'exécution */
/* Déclare les structures et les fonctions de Csound. */
#include "csoundCore.h"
/* Déclare la structure RMP. */
#include "newgen.h"

int rampset (CSOUND *csound, RMP * p) /* à l'initialisation de la note : */
{
    if (*p->icontin == FL(0.0))
        p->curval = *p->istrt; /* reçoit si besoin la nouvelle valeur de début */
    p->vincr = *p->incr / csound->esr; /* fixe l'incrément au taux-s par sec. */
    p->countdown = *p->itime * csound->esr; /* compteur pour iduree en secondes */
    return OK;
}

int ramp (CSOUND *csound, RMP * p) /* pendant l'exécution de la note : */
{
    MYFLT *rsltp = p->result; /* initialise un pointeur sur le tableau de sortie */
    int nn = csound->ksmps; /* taille du tableau donnée par l'orchestre */
    do
    {
        *rsltp++ = p->curval; /* copie la valeur courante vers la sortie */
        if (--p->countdown > 0) /* pour les premières iduree secondes, */
            continue;
    } while (nn-- > 0);
}
```

```
        p->curval += p->vincr;          /* incrémenter la valeur */
    }
    while (--nn);
    return OK;
}
```

Maintenant nous ajoutons ce module à la table du traducteur dans `entry1.c`, sous le nom d'opcode `ramp` :

```
#include "newgen.h"

int rampset(CSOUND *, RMP *), ramp(CSOUND *, RMP *);

/* opname dsblksiz thread outtypes inttypes iopadr kopadr aopadr */
{ "ramp", S(RMP), 5, "a", "iiio", (SUBR)rampset, (SUBR)NULL, (SUBR)ramp },
```

Finalement, il faut relier Csound avec le nouveau module. Ajoutez le nom du fichier C à la liste `libCsoundSources` dans le fichier `SConstruct` :

```
libCsoundSources = Split('''
Engine/auxfd.c
...
OOps/newgen.c
...
Top/utility.c
''')
```

Lancez `scons` comme vous le feriez pour toute autre construction de Csound, et le nouveau module sera intégré dans votre Csound.

Les actions ci-dessus ont ajouté un nouveau générateur au langage Csound. C'est une fonction de rampe linéaire au taux audio qui modifie une valeur d'entrée selon une pente définie par l'utilisateur pour une durée donnée. Cette rampe peut éventuellement continuer depuis la dernière valeur de la note précédente. L'entrée correspondante du manuel de Csound ressemblerait à ceci :

```
ar ramp idebut, ipente, iduree [, icontin]
```

*idebut* -- valeur du début d'une rampe linéaire au taux audio. Eventuellement ignorée s'il y a un drapeau de continuité.

*ipente* -- pente de la rampe, exprimée comme le taux de changement des y par seconde.

*iduree* -- durée de la rampe en secondes, après laquelle la valeur est tenue jusqu'à la fin de la note.

*icontin* (facultatif) -- drapeau de continuité. S'il est à zéro, la rampe démarrera depuis l'entrée *idebut*. Sinon, la rampe démarrera depuis la dernière valeur de la note précédente. La valeur par défaut est zéro.

Le fichier `newgen.h` comprend une liste de paramètres de sortie et d'entrée définie sur une ligne. Ce sont les ports par lesquels le nouveau générateur communiquera avec les autres générateurs dans un instrument. La communication se fait par *adresse*, pas par *valeur*, et c'est une liste de pointeurs sur des valeurs de type MYFLT (*double* si la macro `USE_DOUBLE` est définie, et *float* autrement). Il n'y a aucune restriction sur les noms, mais les types d'argument d'entrée-sortie sont définis plus loin par des chaînes de

caractères dans `entry1.c` (intypes, outypes). Les types intypes sont habituellement *x*, *a*, *k*, et *i*, suivant les conventions normales du manuel de Csound ; on trouve aussi *o* (facultatif, par défaut 0), *p* (facultatif, par défaut 1). Les types outypes comprennent *a*, *k*, *i* et *s* (asig ou ksig). Il est important que tous les noms d'argument de la liste se voient attribuer un type d'argument correspondant dans `entry1.c`. De plus, les arguments de type-*i* ne sont valides qu'à l'initialisation, et les arguments des autres types ne sont valables que pendant l'exécution. Les lignes suivantes de la structure RMP déclarent l'espace de travail nécessaire pour que le code soit réentrant. Ceci permet d'utiliser le module plusieurs fois dans plusieurs copies d'instrument tout en préservant toutes les données.

Le fichier `newgen.c` contient deux sous-programmes, appelés chacun avec un pointeur sur l'instance de Csound et un pointeur sur la structure RMP allouée de façon unique et ses données. Les sous-programmes peuvent être de trois sortes : initialisation de note, génération de signal au taux-*k*, génération de signal au taux-*a*. Normalement, un module requiert deux de ces sous-programmes : initialisation, et un sous-programme soit de taux-*k*, soit de taux-*a* qui sera inséré dans divers listes chaînées de tâches exécutables quand un instrument est activé. Les type de chaînage apparaissent dans `entry1.c` sous deux formes : noms *isub*, *ksub* et *asub* ; et un index de chaînage qui est la somme de *isub*=1, *ksub*=2, *asub*=4. Le code lui-même peut référencer (mais ça ne devrait être qu'en lecture) les membres publiques de la structure CSOUND définie dans `csoundCore.h`, dont les plus utiles sont :

OPARMS	*oparms	
MYFLT	esr	taux d'échantillonnage défini par l'utilisateur
MYFLT	ekr	taux de contrôle défini par l'utilisateur
int	ksmps	ksmps défini par l'utilisateur
int	nchnls	nchnls défini par l'utilisateur
int	oparms->odebug	option -v de la ligne de commande
int	oparms->msglevel	option -m de la ligne de commande
MYFLT	tpidsr	2 * PI / esr

## Tables de Fonction

pour accéder aux tables de fonction en mémoire, une aide spéciale est disponible. La nouvelle structure définie doit comprendre un pointeur

```
FUNC          *ftp;
```

initialisé par l'instruction

```
ftp = csound->FTFind(csound, p->ifuncno);
```

où MYFLT \*ifuncno est un argument d'entrée de type-*i* contenant le numéro de la ftable. La table stockée est alors en `ftp->ftable`, et d'autres données comme sa longueur, les masques de phase, les convertisseurs cps-incrément, sont aussi accessibles depuis ce pointeur. Voir la structure FUNC dans `csoundCore.h`, le code de `csoundFTFind()` dans `fgens.c`, et le code de `oscset()` et de `koscil()` dans `oops/ugens2.c`.

## Espace Supplémentaire

Parfois les besoins en espace d'un module sont trop grands pour faire partie d'une structure (limite supérieure de 65279 octets, due au paramètre en entier court non-signé *dsblksiz* et aux codes réservés >= 0xFF00), ou ils dépendent d'une valeur d'argument-*i* qui n'est pas connue avant l'initialisation. De l'espace supplémentaire peut être alloué dynamiquement et géré proprement en incluant la ligne

```
AUXCH          auxch;
```

dans la structure défini (\*p), puis en utilisant ce type de code dans le module d'initialisation :

```
csound->AuxAlloc(csound, npoints * sizeof(MYFLT), &p->auxch);
```

L'adresse de l'espace auxiliaire est gardée dans une chaîne d'espaces similaires appartenant à cet instrument, et elle est gérée automatiquement lorsque l'instrument est dupliqué ou passé au ramasse-miettes durant l'exécution. L'assignation

```
void *auxp = p->auxch.auxp;
```

trouvera les espaces alloués pour une utilisation pendant l'initialisation et pendant l'exécution. Voir la structure LINSEG dans `ugens1.h` et le code de `lsgset()` and `klseg()` dans `OOps/ugens1.c`.

## Partage de Fichier

Lorsque l'on accède souvent à un fichier externe, ou si on le fait depuis plusieurs endroits, il est souvent efficace de lire le fichier entier dans la mémoire. On accomplit ceci en incluant la ligne

```
MEMFIL      *mfp;
```

dans la structure définie (\*p), puis en utilisant le style de code suivant dans le module d'initialisation :

```
p->mfp = csound->ldmemfile(csound, nomfic);
```

où `char *nomfic` est une chaîne contenant le nom du fichier requis. Les données lues se trouveront entre

```
(char *)p->mfp->beginp; et (char *)p->mfp->endp;
```

Les fichiers chargés n'appartiennent pas à un instrument particulier, mais sont automatiquement partagés pour des accès multiples. Voir la structure ADSYN dans `ugens3.h` et le code de `adset()` et de `adsyn()` dans `OOps/ugens3.c`.

## Arguments Chaîne

Pour permettre un argument d'entrée de type chaîne (disons MYFLT \*`inomfic`) dans votre structure définie (\*p), assignez-lui le type d'argument *S* dans `entry1.c`, et incluez le code suivant dans le module d'initialisation :

```
strcpy(nomfic, (char*)p->inomfic);
```

Voir le code pour `adset()` dans `OOps/ugens3.c`, `lprdset()` dans `OOps/ugens5.c`, et `pvset()` dans `OOps/ugens8.c`.

## Ajouter un Générateur Unitaire comme Plugin

La procédure pour créer un générateur unitaire comme plugin ressemble beaucoup à celle qui est utilisée pour créer un générateur intégré. Le code du générateur unitaire sera le même à part les différences suivantes.

En supposant à nouveau que votre générateur s'appelle `newgen`, effectuez les étapes suivantes :

1. Ecrivez vos fichiers `newgen.c` et `newgen.h` comme vous le feriez pour un générateur unitaire intégré. Mettez ces fichiers dans le répertoire `csound5/Opcodes`.
2. Mettez `#include "csdl.h"` dans les sources de votre générateur unitaire, au lieu de `#include "csoundCore.h"`.
3. Ajoutez vos champs `OENTRY` et les fonctions d'enregistrement du générateur unitaire au bas de votre fichier C. Exemple (mais vous pouvez avoir autant de générateurs unitaires que vous le voulez dans un plugin) :

```
#define S sizeof
static OENTRY localops[] = {
{
    { "rampt", S(RMP), 5, "a", "iiio", (SUBR)rampsset, (SUBR)NULL, (SUBR)ramp },
};
/*
 * La macro suivante de csdl.h définit
 * la fonction d'enregistrement d'opcode "csound_opcode_init()"
 * pour la table des opcodes locaux.
 */
LINKAGE
```

4. Ajoutez votre plugin comme nouvelle cible dans la section des opcodes en plugin du fichier de construction `SConstruct` :

```
pluginEnvironment.SharedLibrary('newgen',
    Split('Opcodes/newgen.c
    Opcodes/un_autre_fichier_utilise_par_newgen.c
    Opcodes/encore_un_autre_fichier_utilise_par_newgen.c'))
```

5. Lancer la construction de Csound de la manière usuelle.

## Référence de `OENTRY`

La structure `OENTRY` (voir `H/csoundCore.h`, `Engine/entry1.c`, et `Engine/rdorich.c`) contient les champs publics suivants :

`opname`, `dsblksiz`, `thread`, `outypes`, `intypes`, `iopadr`, `kopadr`, `aopadr`

`dsblksiz` Il y a deux types d'opcode, polymorphe et non-polymorphe. Pour les opcodes non-polymorphes, le drapeau `dsblksiz` spécifie la taille de la structure de l'opcode en octets, et les arguments sont toujours passés à l'opcode au même taux. Les opcodes polymorphes peuvent accepter des arguments à des taux différents, et la façon dont ces arguments sont réellement distribués aux autres opcodes est déterminée par le drapeau `dsblksiz` et les conventions de nommage suivantes (note : la liste suivante est incomplète, voir `Engine/entry1.c` pour tous les codes spéciaux possibles pour `dsblksiz`) :

0xffff	Le type du premier argument en sortie détermine quelle fonction de générateur unitaire est réellement appelée : <code>xxx -&gt; xxx.a</code> , <code>xxx.i</code> , ou <code>xxx.k</code> .
0xffffe	Les types des deux premiers arguments en entrée déterminent quelle fonction de générateur unitaire est réellement appelée : <code>xxx -&gt; xxx.aa</code> , <code>xxx.ak</code> , <code>xxx.ka</code> , ou <code>xxx.kk</code> , comme dans le générateur unitaire <code>oscil</code> .
0xffffd	Fait référence à un argument en entrée de type <code>a</code> ou <code>k</code> , comme dans le générateur unitaire <code>peak</code> .
thread	Spécifie le(s) taux utilisé(s) pour appeler les fonctions de générateur unitaire, comme suit :

**Tableau 22. Taux d'appel des ugens selon le paramètre thread**

0	taux-i <i>ou</i> taux-k (sortie B seulement)
1	taux-i
2	taux-k
3	taux-i <i>et</i> taux-k
4	taux-a
5	taux-i <i>et</i> taux-a
7	taux-i <i>et</i> (taux-k <i>ou</i> taux-a)

outypes	Liste les valeurs de retour des fonctions de générateur unitaire, s'il y en a. Les types permis sont (note : la liste suivante est incomplète, voir <code>Engine/entry1.c</code> pour tous les types possibles en sortie) :
---------	---

**Tableau 23. Liste des types de sortie des ugens**

i	scalaire de taux-i
k	scalaire de taux-k
a	vecteur de taux-a
x	scalaire de taux-k ou vecteur de taux-a
f	type fsig de flux pvoc de taux-f
m	arguments multiples en sortie de taux-a

intypes	Liste les arguments, s'il y en a, que prennent les fonctions de générateur unitaire. Les types permis sont (note : la liste suivante est incomplète, voir <code>Engine/entry1.c</code> pour tous les types possibles en entrée) :
---------	---

**Tableau 24. Liste des types d'entrée des ugens**

i	scalaire de taux-i
k	scalaire de taux-k
a	vecteur de taux-a
x	scalaire de taux-a ou vecteur de taux-a

f	type fsig de flux pvoc de taux-f
S	Chaîne
B	
l	
m	Commence une liste indéfinie d'arguments de taux-i (n'importe quel nombre)
M	Commence une liste indéfinie d'arguments (n'importe quel taux, n'importe quel nombre)
N	Commence une liste indéfinie d'arguments facultatifs (aux taux-a, -k, -i, ou -s) (n'importe quel nombre impair)
n	Commence une liste indéfinie d'arguments au taux-i (n'importe quel nombre impair)
O	facultatif au taux-k, 0 par défaut
o	facultatif au taux-i, 0 par défaut
p	facultatif au taux-i, 1 par défaut
q	facultatif au taux-i, 10 par défaut
v	facultatif au taux-k, 0.5 par défaut
v	facultatif au taux-i, 0.5 par défaut
j	facultatif au taux-i, -1 par défaut
h	facultatif au taux-i, 127 par défaut
y	Commence une liste indéfinie d'arguments au taux-a (n'importe quel nombre)
z	Commence une liste indéfinie d'arguments au taux-k (n'importe quel nombre)
Z	Commence une liste indéfinie d'argumenents alternant les taux-k et -a (kaka...) (n'importe quel nombre)

iopadr      L'adresse de la fonction du générateur unitaire (de type `int (*SUBR)(CSOUND *, void *)`) qui est appelée à l'initialisation, ou NULL s'il n'y a pas de fonction.

kopadr      L'adresse de la fonction du générateur unitaire (de type `int (*SUBR)(CSOUND *, void *)`) qui est appelée au taux-k, ou NULL s'il n'y a pas de fonction.

aopadr      L'adresse de la fonction du générateur unitaire (de type `int (*SUBR)(CSOUND *, void *)`) qui est appelée au taux-a, ou NULL s'il n'y a pas de fonction.



---

## **Partie IV. Référence Rapide des Opcodes**

---

---

## Table des matières

Référence Rapide des Opcodes .....	2517
------------------------------------	------

---

# Référence Rapide des Opcodes

## Syntaxe de l'Orchestre : En-tête.

```
0dbfs = iarg
0dbfs

kr = iarg

ksmps = iarg

nchnls = iarg

sr = iarg
```

## Syntaxe de l'Orchestre : Bloc d'Instructions.

```
endin

endop

instr i, j, ...

opcode nom, outtypes, intypes
```

## Syntaxe de l'Orchestre : Macros.

```
#define NAME # replacement text #

#define NAME(a' b' c') # replacement text #

$NAME

#ifdef NAME
....
#else
....
#endif

#ifndef NAME
....
#else
....
#endif

#include "filename"

#undef NAME
```

## Générateurs de Signal : Synthèse/Resynthèse Additive.

```
ares adsyn kamod, kfmod, ksmod, ifilcod

ares adsynt kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]

ar adsynt2 kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

```
ares hsboscil kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn \
[, ioctcnt] [, iphs]
```

### Générateurs de Signal : Oscillateurs Élémentaires.

```
kres lfo kamp, kcps [, itype]
ares lfo kamp, kcps [, itype]

ares oscbnk kcps, kamd, kfmd, kpmd, iovrlap, iseed, kllminf, kllmaxf, \
kl2minf, kl2maxf, ilfomode, kegminf, kegmaxf, kegminl, kegmaxl, \
kegming, kegmaxq, iegmode, kfn [, il1fn] [, il2fn] [, iegffn] \
[, ieg1fn] [, iegqfn] [, itabl] [, ioutfn]

ares oscil xamp, xcps, ifn [, iphs]
kres oscil kamp, kcps, ifn [, iphs]

ares oscil3 xamp, xcps, ifn [, iphs]
kres oscil3 kamp, kcps, ifn [, iphs]

ares oscili xamp, xcps, ifn [, iphs]
kres oscili kamp, kcps, ifn [, iphs]

ares oscilikt xamp, xcps, kfn [, iphs] [, istor]
kres oscilikt kamp, kcps, kfn [, iphs] [, istor]

ares osciliktp kcps, kfn, kphs [, istor]

ares oscilikts xamp, xcps, kfn, async, kphs [, istor]

ares osciln kamp, ifrq, ifn, itimes

ares oscils iamp, icps, iphs [, iflg]

ares poscil aamp, acps, ifn [, iphs]
ares poscil aamp, kcps, ifn [, iphs]
ares poscil kamp, acps, ifn [, iphs]
ares poscil kamp, kcps, ifn [, iphs]
ires poscil kamp, kcps, ifn [, iphs]
kres poscil kamp, kcps, ifn [, iphs]

ares poscil3 kamp, kcps, ifn [, iphs]
kres poscil3 kamp, kcps, ifn [, iphs]

kout vibr kAverageAmp, kAverageFreq, ifn

kout vibrato kAverageAmp, kAverageFreq, kRandAmountAmp, \
kRandAmountFreq, kAmpMinRate, kAmpMaxRate, kcpsMinRate, \
kcpsMaxRate, ifn [, iphs]
```

### Générateurs de Signal : Oscillateurs à Spectre Dynamique.

```
ares buzz xamp, xcps, knh, ifn [, iphs]

ares gbuzz xamp, xcps, knh, klh, kmul, ifn [, iphs]

ares mpulse kamp, kintvl [, ioffset]

ares vco xamp, xcps, iwave, kpw [, ifn] [, imaxd] [, ileak] [, inyx] \
[, iphs] [, iskip]

ares vco2 kamp, kcps [, imode] [, kpw] [, kphs] [, inyx]

kfn vco2ft kcps, iwave [, inyx]

ifn vco2ift icps, iwave [, inyx]
```

ifn **vco2init** iwave [, ibasfn] [, ipmul] [, iminsiz] [, imaxsiz] [, isrcft]

### Générateurs de Signal : Synthèse FM.

```
al, a2 crossfm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
al, a2 crossfmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
al, a2 crosspm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
al, a2 crosspmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
al, a2 crossfmpm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
al, a2 crossfmpmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
```

```
ares fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
    ifn4, ivfn
```

```
ares fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
    ifn4, ivfn
```

```
ares fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
    ifn4, ivfn
```

```
ares fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \
    ifn3, ifn4, ivfn
```

```
ares fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \
    ifn3, ifn4, ivfn
```

```
ares fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, \
    ifn2, ifn3, ifn4, ivibfn
```

```
ares fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
    ifn4, ivfn
```

```
ares foscil xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

```
ares foscili xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

### Générateurs de Signal : Synthèse Granulaire.

```
asig diskgrain Sfname, kamp, kfreq, kpitch, kgrsize, kprate, \
    ifun, iolaps [, imaxgrsize, ioffset]
```

```
ares fof xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
    ifna, ifnb, itotdur [, iphs] [, ifmode] [, iskip]
```

```
ares fof2 xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
    ifna, ifnb, itotdur, kphs, kgliss [, iskip]
```

```
ares fog xamp, xdens, xtrans, aspd, koct, kband, kris, kdur, kdec, \
    iolaps, ifna, ifnb, itotdur [, iphs] [, itmode] [, iskip]
```

```
ares grain xamp, xpitch, xdens, kampoff, kpitchoff, kgdur, igfn, \
    iwfn, imgdur [, igrnd]
```

```
ares grain2 kcps, kfmd, kgdur, iovrlp, kfn, iwfn [, irpow] \
    [, iseed] [, imode]
```

```
ares grain3 kcps, kphs, kfmd, kpm, kgdur, kdens, imaxovr, kfn, iwfn, \
    kfrpow, kprpow [, iseed] [, imode]
```

```
ares granule xamp, ivoice, iratio, imode, ithd, ifn, ipshift, igskip, \
    igskip_os, ilength, kgap, igap_os, kgsz, igsz_os, iatt, idec \
    [, iseed] [, ipitch1] [, ipitch2] [, ipitch3] [, ipitch4] [, ifnenv]
```

```
al [, a2, a3, a4, a5, a6, a7, a8] partikkel agrainfreq, \
    kdistribution, idisttab, async, kenv2amt, ienv2tab, ienv_attack, \
    ienv_decay, ksustain_amount, ka_d_ratio, kduration, kamp, igainmask, \
    kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
```

```

        ifmampstab, kfmenv, icosine, ktraincps, knumpartials, kchroma, \
        ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, \
        kwaveform4, iwaveampstab, asamplepos1, asamplepos2, asamplepos3, \
        asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains \
        [, iopcode_id]

async [,aphase] partikkelsync iopcode_id

ares [, ac] sndwarp xamp, xtimewarp, xresample, ifn1, ibeg, iwsiz, \
        irandw, ioverlap, ifn2, itimemode

arl, ar2 [,ac1] [, ac2] sndwarpst xamp, xtimewarp, xresample, ifn1, \
        ibeg, iwsiz, irandw, ioverlap, ifn2, itimemode

asig syncgrain kamp, kfreq, kpitch, kgrsize, kprate, ifun1, \
        ifun2, iolaps

asig syncloop kamp, kfreq, kpitch, kgrsize, kprate, klstart, \
        klend, ifun1, ifun2, iolaps[,istart, iskip]

ar vosim kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]

```

### Générateurs de Signal : Synthèse Hyper Vectorielle.

```

hvs1 kx, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]

hvs2 kx, ky, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]

hvs3 kx, ky, kz, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfig-
Tab]

```

### Générateurs de Signal : Générateurs Linéaires et Exponentiels.

```

kout expcurve kindex, ksteepness

ares expon ia, idur, ib
kres expon ia, idur, ib

ares expseg ia, idur1, ib [, idur2] [, ic] [...]
kres expseg ia, idur1, ib [, idur2] [, ic] [...]

ares expsega ia, idur1, ib [, idur2] [, ic] [...]

ares expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
kres expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz

kout gainslider kindex

ares jspline xamp, kcpsMin, kcpsMax
kres jspline kamp, kcpsMin, kcpsMax

ares line ia, idur, ib
kres line ia, idur, ib

ares linseg ia, idur1, ib [, idur2] [, ic] [...]
kres linseg ia, idur1, ib [, idur2] [, ic] [...]

ares linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
kres linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz

kout logcurve kindex, ksteepness

ksig loopseg kfreq, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
        [, ktime2] [, kvalue2] [...]

ksig loopsegp kphase, kvalue0, kdur0, kvalue1 \
        [, kdur1, ... , kdurN-1, kvalueN]

```

```
ksig looptseg kfreq, ktrig, ktime0, kvalue0, ktype, [, ktime1] [,ktype1] [, kvalue1] \
    [, ktime2] [,ktype2] [, kvalue2] [...]
```

```
ksig loopxseg kfreq, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
    [, ktime2] [, kvalue2] [...]
```

```
ksig lpshold kfreq, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
    [, ktime2] [, kvalue2] [...]
```

```
ksig lpsholdp kphase, ktrig, ktime0, kvalue0 [, ktime1] [, kvalue1] \
    [, ktime2] [, kvalue2] [...]
```

```
ares rspline xrangeMin, xrangeMax, kcpsMin, kcpsMax
kres rspline krangeMin, krangeMax, kcpsMin, kcpsMax
```

```
kscl scale kinput, kmax, kmin
```

```
ares transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
kres transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

```
ares transegr ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
kres transegr ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

### Générateurs de Signal : Générateurs d'Enveloppe.

```
ares adsr iatt, idec, islev, irel [, idel]
kres adsr iatt, idec, islev, irel [, idel]
```

```
ares envlpx xamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]
kres envlpx kamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]
```

```
ares envlpxr xamp, irise, idec, ifn, iatss, iatdec [, ixmod] [,irind]
kres envlpxr kamp, irise, idec, ifn, iatss, iatdec [, ixmod] [,irind]
```

```
ares linen xamp, irise, idur, idec
kres linen kamp, irise, idur, idec
```

```
ares linenr xamp, irise, idec, iatdec
kres linenr kamp, irise, idec, iatdec
```

```
ares madsr iatt, idec, islev, irel [, idel] [, ireltim]
kres madsr iatt, idec, islev, irel [, idel] [, ireltim]
```

```
ares mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
kres mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
```

```
ares xadsr iatt, idec, islev, irel [, idel]
kres xadsr iatt, idec, islev, irel [, idel]
```

### Générateurs de Signal : Modèles et Emulations.

```
ares bamboo kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]
```

```
ares barmodel kbcL, kbcR, iK, ib, kscan, iT30, ipos, ivel, iwid
```

```
ares cabasa iamp, idettack [, inum] [, idamp] [, imaxshake]
```

```
aI3, aV2, aV1 chuap kL, kR0, kC1, kG, kGa, kGb, kE, kC2, iI3, iV2, iV1, ktime_step
```

```
ares crunch iamp, idettack [, inum] [, idamp] [, imaxshake]
```

```
ares dripwater kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]
```

```
ares gogobel kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivfn
```

```

ares guiro kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] [, ifreq1]
ax, ay, az lorenz ksv, krv, kbv, kh, ix, iy, iz, iskip [, iskipinit]
kiter, koutrig mandel ktrig, kx, ky, kmaxIter
ares mandol kamp, kfreq, kpluck, kdetune, kgain, ksize, ifn [, iminfreq]
ares marimba kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec \
    [, idoubles] [, itriples]
ares moog kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn
ax, ay, az planet kmass1, kmass2, ksep, ix, iy, iz, ivx, ivy, ivz, idelta \
    [, ifriction] [, iskip]
ares prepiano ifreq, iNS, iD, iK, \
    iT30, iB, kbcl, kbcr, imass, ifreq, iinit, ipos, ivel, isfreq, \
    isspread[, irattles, irubbers]
al,ar prepiano ifreq, iNS, iD, iK, \
    iT30, iB, kbcl, kbcr, imass, ifreq, iinit, ipos, ivel, isfreq, \
    isspread[, irattles, irubbers]
ares sandpaper iamp, idettack [, inum] [, idamp] [, imaxshake]
ares sekere iamp, idettack [, inum] [, idamp] [, imaxshake]
ares shaker kamp, kfreq, kbeans, kdamp, ktimes [, idecay]
ares sleighbells kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]
ares stix iamp, idettack [, inum] [, idamp] [, imaxshake]
ares tambourine kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]
ares vibes kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec
ares voice kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn

```

### Générateurs de Signal : Phaseurs.

```

ares phasor xcps [, iphs]
kres phasor kcps [, iphs]

ares phasorbnk xcps, kndx, icnt [, iphs]
kres phasorbnk kcps, kndx, icnt [, iphs]

aphase, asyncout syncphasor xcps, asyncin, [, iphs]

```

### Générateurs de Signal : Générateurs de Nombres Aléatoires (de Bruit).

```

ares betarand krange, kalpha, kbeta
ires betarand krange, kalpha, kbeta
kres betarand krange, kalpha, kbeta

ares bexprnd krange
ires bexprnd krange
kres bexprnd krange

ares cauchy kalpha
ires cauchy kalpha
kres cauchy kalpha

aout cuserrnd kmin, kmax, ktableNum
iout cuserrnd imin, imax, itableNum

```



```

kout cuserrnd kmin, kmax, ktableNum

aout duserrnd ktableNum
iout duserrnd itableNum
kout duserrnd ktableNum

ares exprand klambda
ires exprand klambda
kres exprand klambda

ares gauss krange
ires gauss krange
kres gauss krange

kout jitter kamp, kcpsMin, kcpsMax

kout jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, kamp3, kcps3

ares linrand krange
ires linrand krange
kres linrand krange

ares noise xamp, kbeta

ares pcauchy kalpha
ires pcauchy kalpha
kres pcauchy kalpha

ares pinkish xin [, imethod] [, inumbands] [, iseed] [, iskip]

ares poisson klambda
ires poisson klambda
kres poisson klambda

ares rand xamp [, iseed] [, isel] [, ioffset]
kres rand xamp [, iseed] [, isel] [, ioffset]

ares randh xamp, xcps [, iseed] [, isize] [, ioffset]
kres randh kamp, kcps [, iseed] [, isize] [, ioffset]

ares randi xamp, xcps [, iseed] [, isize] [, ioffset]
kres randi kamp, kcps [, iseed] [, isize] [, ioffset]

ares random kmin, kmax
ires random imin, imax
kres random kmin, kmax

ares randomh kmin, kmax, acps
kres randomh kmin, kmax, kcps

ares randomi kmin, kmax, acps
kres randomi kmin, kmax, kcps

ax rnd31 kscl, krpow [, iseed]
ix rnd31 iscl, irpow [, iseed]
kx rnd31 kscl, krpow [, iseed]

seed ival

kout trandom ktrig, kmin, kmax

ares trirand krange
ires trirand krange
kres trirand krange

ares unirand krange
ires unirand krange
kres unirand krange

aout = urd(ktableNum)
iout = urd(itableNum)
kout = urd(ktableNum)

ares weibull ksigma, ktau
ires weibull ksigma, ktau

```

kres **weibull** ksigma, ktau

## Générateurs de Signal : Reproduction de Sons Echantillonnés.

```

a1 bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats \
    [, istutterspeed] [, istutterchance] [, ienvchoice ]

a1,a2 bbcuts asource1, asource2, ibps, isubdiv, ibarlength, iphrasebars, \
    inumrepeats [, istutterspeed] [, istutterchance] [, ienvchoice]

asig flooper kamp, kpitch, istart, idur, ifad, ifn

asig flooper2 kamp, kpitch, kloopstart, kloopend, kcrossfade, ifn \
    [, istart, imode, ifenv, iskip]

aleft, aright fluidAllOut

fluidCCi iEngineNumber, iChannelNumber, iControllerNumber, iValue

fluidCCK iEngineNumber, iChannelNumber, iControllerNumber, kValue

fluidControl ienginenum, kstatus, kchannel, kdata1, kdata2

ienginenum fluidEngine [iReverbEnabled] [, iChorusEnabled] [, iNumChannels] [, iPolyphony]

isfnum fluidLoad soundfont, ienginenum[, ilistpresets]

fluidNote ienginenum, ichannelnum, imidikey, imidivel

aleft, aright fluidOut ienginenum

fluidProgramSelect ienginenum, ichannelnum, isfnum, ibanknum, ipresetnum

fluidSetInterpMethod ienginenum, ichannelnum, iInterpMethod

ar1 [,ar2] loscil xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
    [, imod2] [, ibeg2] [, iend2]

ar1 [,ar2] loscil3 xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
    [, imod2] [, ibeg2] [, iend2]

ar1 [, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
    ar15, ar16] loscilx xamp, kcps, ifn \
    [, iwsiz, ibas, istr, imod1, ibeg1, iend1]

ares lphasor xtrns [, ilps] [, ilpe] [, imode] [, istr] [, istor]

ares lposcil kamp, kfregratio, kloop, kend, ifn [, iphs]

ares lposcil3 kamp, kfregratio, kloop, kend, ifn [, iphs]

ar lposcila aamp, kfregratio, kloop, kend, ift [, iphs]

ar1, ar2 lposcilsa aamp, kfregratio, kloop, kend, ift [, iphs]

ar1, ar2 lposcilsa2 aamp, kfregratio, kloop, kend, ift [, iphs]

sfilist ifilhandle

ar1, ar2 sfinstr ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
    [, iflag] [, ioffset]

ar1, ar2 sfinstr3 ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
    [, iflag] [, ioffset]

ares sfinstr3m ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
    [, iflag] [, ioffset]

```

```
ares sfinstrm ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
    [, iflag] [, ioffset]

ir sfload "filename"

ar1, ar2 sflooper ivel, inotenum, kamp, kpitch, ipreindex, kloopstart, kloopend,
kcrossfade, ifn \
    [, istart, imode, ifenv, iskip]

sfpassign istartindex, ifilhandle[, imsgs]

ar1, ar2 sfplay ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

ar1, ar2 sfplay3 ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

ares sfplay3m ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

ares sfplaym ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

sfplist ifilhandle

ir sfpreset iprog, ibank, ifilhandle, ipreindex

asig, krec sndloop ain, kpitch, ktrig, idur, ifad

ares waveset ain, krep [, ilen]
```

### Générateurs de Signal : Synthèse par Balayage.

```
scanhammer isrc, idst, ipos, imult

ares scans kamp, kfreq, ifn, id [, iorder]

aout scantable kamp, kpch, ipos, imass, istiff, idamp, ivel

scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
    kstif, kcentr, kdamp, ileft,  iright, kpos, kstrngth, ain, idisp, id

kpos, kvel xscanmap iscan, kamp, kvamp [, iwhich]

ares xscans kamp, kfreq, ifntraj, id [, iorder]

xscansmap kpos, kvel, iscan, kamp, kvamp [, iwhich]

xscanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
    kstif, kcentr, kdamp, ileft,  iright, kpos, kstrngth, ain, idisp, id
```

### Générateurs de Signal : Accès aux Tables.

```
kres oscill idel, kamp, idur, ifn

kres oscilli idel, kamp, idur, ifn

ir tab_i indx, ifn[, ixmode]
kr tab kndx, ifn[, ixmode]
ar tab xndx, ifn[, ixmode]
tabw_i isig, indx, ifn [,ixmode]
tabw ksig, kndx, ifn [,ixmode]
tabw asig, andx, ifn [,ixmode]

ares table andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires table indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres table kndx, ifn [, ixmode] [, ixoff] [, iwrap]

ares table3 andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires table3 indx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
kres table3 kndx, ifn [, ixmode] [, ixoff] [, iwrap]

ares tablei andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires tablei indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres tablei kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

### Générateurs de Signal : Synthèse par Terrain d'Ondes.

```
aout wterrain kamp, kpch, k_xcenter, k_ycenter, k_xradius, k_yradius, \
      itabx, itaby
```

### Générateurs de Signal : Modèles Physiques par Guide d'Onde.

```
ares pluck kamp, kcps, icps, ifn, imeth [, iparm1] [, iparm2]

ares repluck iplk, kamp, icps, kpick, krefl, axcite

ares streson asig, kfrq, ifdbgain

ares wgbow kamp, kfreq, kpres, krat, kvibf, kvamp, ifn [, iminfreq]

ares wgbowedbar kamp, kfreq, kpos, kbowpres, kgain [, iconst] [, itvel] \
      [, ibowpos] [, ilow]

ares wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp, ifn [, iminfreq]

ares wgclar kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, ifn \
      [, iminfreq]

ares wgflute kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, ifn \
      [, iminfreq] [, ijetrf] [, iendrf]

ares wgpluck icps, iamp, kpick, iplk, idamp, ifilt, axcite

ares wgpluck2 iplk, kamp, icps, kpick, krefl
```

### E/S de Signal : E/S Fichier.

```
dumpk ksig, ifilename, iformat, iprd

dumpk2 ksig1, ksig2, ifilename, iformat, iprd

dumpk3 ksig1, ksig2, ksig3, ifilename, iformat, iprd

dumpk4 ksig1, ksig2, ksig3, ksig4, ifilename, iformat, iprd

ficlose ihandle
ficlose Sfilename

fin ifilename, iskipframes, iformat, ain1 [, ain2] [, ain3] [, ...]

fini ifilename, iskipframes, iformat, in1 [, in2] [, in3] [, ...]

fink ifilename, iskipframes, iformat, kin1 [, kin2] [, kin3] [, ...]

ihandle fiopen ifilename, imode

fout ifilename, iformat, aout1 [, aout2, aout3, ..., aoutN]

fouti ihandle, iformat, iflag, iout1 [, iout2, iout3, ..., ioutN]
```

```

foutir ihandle, iformat, iflag, iout1 [, iout2, iout3, ..., ioutN]

foutk ifilename, iformat, kout1 [, kout2, kout3, ..., koutN]

fprintks "filename", "string", [, kval1] [, kval2] [...]

fprints "filename", "string" [, ival1] [, ival2] [...]

kres readk ifilename, iformat, iprd

kr1, kr2 readk2 ifilename, iformat, iprd

kr1, kr2, kr3 readk3 ifilename, iformat, iprd

kr1, kr2, kr3, kr4 readk4 ifilename, iformat, iprd

```

### E/S de Signal : Entrée de Signal.

```

ar1 [, ar2 [, ar3 [, ... ar24]]] diskin ifilcod, kpitch [, iskiptim] \
    [, iwraparound] [, iformat] [, iskipinit]

a1[, a2[, ... a24]] diskin2 ifilcod, kpitch[, iskiptim \
    [, iwrap[, iformat [, iwsizel[, ibufsize[, iskipinit]]]]]]

ar1 in

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
    ar15, ar16, ar17, ar18, ar19, ar20, ar21, ar22, ar23, ar24, ar25, ar26, \
    ar27, ar28, ar29, ar30, ar31, ar32 in32

ain inch kchan

ar1, ar2, ar3, ar4, ar5, ar6 inh

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8 ino

ar1, ar2, ar3, a4 inq

inrg kstart, ain1 [, ain2, ain3, ..., ainN]

ar1, ar2 ins

kvalue invalue "channel name"
Sname invalue "channel name"

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, \
    ar13, ar14, ar15, ar16 inx

inz ksig1

ar1, ar2 mp3in ifilcod, iskiptim, iformat, iskipinit, ibufsize

ar1[, ar2[, ar3[, ... a24]]] soundin ifilcod [, iskiptim] [, iformat] \
    [, iskipinit] [, ibufsize]

```

### E/S de Signal : Sortie de Signal.

```

mdelay kstatus, kchan, kd1, kd2, kdelay

aout1 [, aout2 ... aoutX] monitor

out asig

out32 asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, asig10, \
    asig11, asig12, asig13, asig14, asig15, asig16, asig17, asig18, \
    asig19, asig20, asig21, asig22, asig23, asig24, asig25, asig26, \

```

```

    asig27, asig28, asig29, asig30, asig31, asig32

outc asig1 [, asig2] [...]

outch kchan1, asig1 [, kchan2] [, asig2] [...]

outh asig1, asig2, asig3, asig4, asig5, asig6

outo asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8

outq asig1, asig2, asig3, asig4

outq1 asig

outq2 asig

outq3 asig

outq4 asig

outrg kstart, aout1 [,aout2, aout3, ..., aoutN]

outs asig1, asig2

outs1 asig

outs2 asig

outvalue "channel name", kvalue
outvalue "channel name", "string"

outx asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, \
    asig9, asig10, asig11, asig12, asig13, asig14, asig15, asig16

outz ksig1

soundout asig1, ifilcod [, iformat]

soundouts asig1, asigr, ifilcod [, iformat]

```

## E/S de Signal : Bus Logiciel.

```

kval chani kchan
aval chani kchan

chano kval, kchan
chano aval, kchan

chn_k Sname, imode[, itype, idflt, imin, imax]
chn_a Sname, imode
chn_s Sname, imode

chnclear Sname

gival chnexport Sname, imode[, itype, idflt, imin, imax]
gkval chnexport Sname, imode[, itype, idflt, imin, imax]
gaval chnexport Sname, imode
gSval chnexport Sname, imode

ival chnget Sname
kval chnget Sname
aval chnget Sname
Sval chnget Sname

chnmix aval, Sname

itype, imode, ictltype, idflt, imin, imax chnparams

ival chnrecv Sname
kval chnrecv Sname

```

```

aval chnrecv Sname
Sval chnrecv Sname

chnsend ival, Sname
chnsend kval, Sname
chnsend aval, Sname
chnsend Sval, Sname

chnset ival, Sname
chnset kval, Sname
chnset aval, Sname
chnset Sval, Sname

setksmps iksmps

xinarg1 [, xinarg2] ... [xinargN] xin

xout xoutarg1 [, xoutarg2] ... [, xoutargN]
```

### E/S de Signal : Impression et Affichage.

```

dispffft xsig, iprd, iwsiz [, iwtyp] [, idbout] [, iwtflg]

display xsig, iprd [, inprds] [, iwtflg]

flashtxt iwhich, String

print iarg [, iarg1] [, iarg2] [...]

printf_i Sfmt, itrig, [iarg1[, iarg2[, ... ]]]
printf Sfmt, ktrig, [xarg1[, xarg2[, ... ]]]

printk itime, kval [, ispace]

printk2 kvar [, inumspaces]

printks "string", itime [, kval1] [, kval2] [...]

prints "string" [, kval1] [, kval2] [...]
```

### E/S de Signal : Requêtes sur les Fichiers Sons.

```

ir filebit ifilcod [, iallowraw]

ir filelen ifilcod, [iallowraw]

ir filenchnls ifilcod [, iallowraw]

ir filepeak ifilcod [, ichnl]

ir filesr ifilcod [, iallowraw]
```

### Modificateurs de Signal : Modificateurs d'Amplitude.

```

ares balance asig, acomp [, ihp] [, iskip]

ares clip asig, imeth, ilimit [, iarg]

ar compress aasig, acsig, kthresh, kloknee, khiknee, kratio, katt, krel, ilook

ares dam asig, kthreshold, icomp1, icomp2, irtime, iftime
```

```
ares gain asig, krms [, ihp] [, iskip]
```

### Modificateurs de Signal : Convolution et Morphing.

```
arl [, ar2] [, ar3] [, ar4] convolve ain, ifilcod [, ichannel]

ares cross2 ain1, ain2, isize, ioverlap, iwin, kbias

ares dconv asig, isize, ifn

a1[, a2[, a3[, ... a8]]] ftconv ain, ift, iplen[, iskip samples \
    [, iirlen[, iskipinit]]]

ftmorf kftndx, iftn, iresfn

arl [, ar2] [, ar3] [, ar4] pconvolve ain, ifilcod [, ipartitions size, ichannel]
```

### Modificateurs de Signal : Retard.

```
ares delay asig, idlt [, iskip]

ares delay1 asig [, iskip]

kr delayk ksig, idel[, imodel]
kr vdelay_k ksig, kdel, imdel[, imodel]

ares delayr idlt [, iskip]

delayw asig

ares deltap kdlt

ares deltap3 xdlt

ares deltapi xdlt

ares deltapn xnumsamps

aout deltapx adel, iws size

deltapxw ain, adel, iws size

ares multitap asig [, itime1] [, igain1] [, itime2] [, igain2] [...]

ares vdelay asig, adel, imaxdel [, iskip]

ares vdelay3 asig, adel, imaxdel [, iskip]

aout vdelayx ain, adl, imd, iws [, ist]

aout1, aout2, aout3, aout4 vdelayxq ain1, ain2, ain3, ain4, adl, imd, iws [, ist]

aout1, aout2 vdelayxs ain1, ain2, adl, imd, iws [, ist]

aout vdelayxw ain, adl, imd, iws [, ist]

aout1, aout2, aout3, aout4 vdelayxwq ain1, ain2, ain3, ain4, adl, \
    imd, iws [, ist]

aout1, aout2 vdelayxws ain1, ain2, adl, imd, iws [, ist]
```



**Modificateurs de Signal : Panning et Spatialisation.**

```
aol, ao2 bformdec isetup, aw, ax, ay, az [, ar, as, at, au, av \
[, abk, al, am, an, ao, ap, aq]]
aol, ao2, ao3, ao4 bformdec isetup, aw, ax, ay, az [, ar, as, at, \
au, av [, abk, al, am, an, ao, ap, aq]]
aol, ao2, ao3, ao4, ao5 bformdec isetup, aw, ax, ay, az [, ar, as, \
at, au, av [, abk, al, am, an, ao, ap, aq]]
aol, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec isetup, aw, ax, ay, az \
[, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]

aol, ao2 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, au, av \
[, abk, al, am, an, ao, ap, aq]]
aol, ao2, ao3, ao4 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, \
au, av [, abk, al, am, an, ao, ap, aq]]
aol, ao2, ao3, ao4, ao5 bformdec1 isetup, aw, ax, ay, az [, ar, as, \
at, au, av [, abk, al, am, an, ao, ap, aq]]
aol, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec1 isetup, aw, ax, ay, az \
[, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]

aw, ax, ay, az bformenc asig, kalpha, kbeta, kord0, kord1
aw, ax, ay, az, ar, as, at, au, av bformenc asig, kalpha, kbeta, \
kord0, kord1, kord2
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc \
asig, kalpha, kbeta, kord0, kord1, kord2, kord3

aw, ax, ay, az bformenc1 asig, kalpha, kbeta
aw, ax, ay, az, ar, as, at, au, av bformenc1 asig, kalpha, kbeta
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1 \
asig, kalpha, kbeta

aleft, aright hrtfer asig, kaz, kelev, HRTFcompact

aleft, aright hrtfmove asrc, kAz, kElev, ifilel, ifiler [, imode, ifade, isr]

aleft, aright hrtfmove2 asrc, kAz, kElev, ifilel, ifiler [, ioverlap, iradius, isr]

aleft, aright hrtfstat asrc, iAz, iElev, ifilel, ifiler [, iradius, isr]

a1, a2 locsend
a1, a2, a3, a4 locsend

a1, a2 locsig asig, kdegree, kdistance, kreverbsend
a1, a2, a3, a4 locsig asig, kdegree, kdistance, kreverbsend

a1, a2, a3, a4 pan asig, kx, ky, ifn [, imode] [, ioffset]

a1, a2 pan2 asig, xp [, imode]

a1, a2, a3, a4 space asig, ifn, ktime, kreverbsend, kx, ky

aW, aX, aY, aZ spat3d ain, kX, kY, kZ, idist, ift, imode, imdel, iovr [, istor]

aW, aX, aY, aZ spat3di ain, iX, iY, iZ, idist, ift, imode [, istor]

spat3dt ioutft, iX, iY, iZ, idist, ift, imode, irlen [, iftnocl]

k1 spdist ifn, ktime, kx, ky

a1, a2, a3, a4 spsend

ar1, ..., ar16 vbap16 asig, kazim [, kelev] [, kspread]

ar1, ..., ar16 vbap16move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]

ar1, ar2, ar3, ar4 vbap4 asig, kazim [, kelev] [, kspread]

ar1, ar2, ar3, ar4 vbap4move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]
```

```

ar1, ..., ar8 vbap8 asig, kazim [, kelev] [, kspread]

ar1, ..., ar8 vbap8move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]

vbaplsinit idim, ilsnum [, idir1] [, idir2] [...] [, idir32]

vbapz inumchnls, istartndx, asig, kazim [, kelev] [, kspread]

vbapzmove inumchnls, istartndx, asig, idur, ispread, ifldnum, ifld1, \
ifld2, [...]

```

### Modificateurs de Signal : Réverbération.

```

ares alpass asig, krvt, ilpt [, iskip] [, insmps]

a1, a2 babo asig, ksrcx, ksrcy, ksrcz, irx, iry, irz [, idiff] [, ifno]

ares comb asig, krvt, ilpt [, iskip] [, insmps]

aoutL, aoutR freeverb ainL, ainR, kRoomSize, kHFDamp[, iSRate[, iSkip]]

ares nestedap asig, imode, imaxdel, idel1, igain1 [, idel2] [, igain2] \
[, idel3] [, igain3] [, istor]

ares nreverb asig, ktime, khdif [, iskip] [, inumCombs] [, ifnCombs] \
[, inumAlpas] [, ifnAlpas]

ares reverb asig, krvt [, iskip]

ares reverb2 asig, ktime, khdif [, iskip] [, inumCombs] \
[, ifnCombs] [, inumAlpas] [, ifnAlpas]

aoutL, aoutR reverb3 ainL, ainR, kfblvl, kfco[, israte[, ipitchm[, iskip]]]

ares valpass asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]

ares vcomb asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]

```

### Modificateurs de Signal : Opérateurs du Niveau Echantillon.

```

denorm a1[, a2[, a3[, ... ]]]

ares diff asig [, iskip]
kres diff ksig [, iskip]

kres downsamp asig [, iwlen]

ares fold asig, kincr

ares integ asig [, iskip]
kres integ ksig [, iskip]

ares interp ksig [, iskip] [, imode]

ares ntrpol asig1, asig2, kpoint [, imin] [, imax]
ires ntrpol isig1, isig2, ipoint [, imin] [, imax]
kres ntrpol ksig1, ksig2, kpoint [, imin] [, imax]

a(x) (arguments de taux-k seulement)

i(x) (arguments de taux-k seulement)

k(x) (arguments de taux-i seulement)

```

```
ares samphold asig, agate [, ival] [, ivstor]
kres samphold ksig, kgate [, ival] [, ivstor]

ares upsamp ksig

kval valet kndx, avar

vaset kval, kndx, avar
```

### Modificateurs de Signal : Limiteurs de Signal.

```
ares limit asig, klow, khigh
ires limit isig, ilow, ihigh
kres limit ksig, klow, khigh

ares mirror asig, klow, khigh
ires mirror isig, ilow, ihigh
kres mirror ksig, klow, khigh

ares wrap asig, klow, khigh
ires wrap isig, ilow, ihigh
kres wrap ksig, klow, khigh
```

### Modificateurs de Signal : Effets Spéciaux.

```
ar distort asig, kdist, ifn[, ihp, istor]

ares distort1 asig, kpregain, kpostgain, kshape1, kshape2[, imode]

ares flanger asig, adel, kfeedback [, imaxd]

ares harmon asig, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, imode, \
    iminfrq, iprd

ares harmon2 asig, koct, kfrq1, kfrq2, icpsmode, ilowest[, ipolarity]
ares harmon3 asig, koct, kfrq1, \
    kfrq2, kfrq3, icpsmode, ilowest[, ipolarity]
ares harmon4 asig, koct, kfrq1, \
    kfrq2, kfrq3, kfrq4, icpsmode, ilowest[, ipolarity]

ares phaser1 asig, kfreq, kord, kfeedback [, iskip]

ares phaser2 asig, kfreq, kq, kord, kmode, ksep, kfeedback
```

### Modificateurs de Signal : Filtres Standard.

```
ares atone asig, khp [, iskip]

ares atonex asig, khp [, inumlayer] [, iskip]

ares biquad asig, kb0, kb1, kb2, ka0, ka1, ka2 [, iskip]

ares biquada asig, ab0, ab1, ab2, aa0, aa1, aa2 [, iskip]

ares butbp asig, kfreq, kband [, iskip]

ares butbr asig, kfreq, kband [, iskip]

ares buthp asig, kfreq [, iskip]

ares butlp asig, kfreq [, iskip]
```

```
ares butterbp asig, kfreq, kband [, iskip]
ares butterbr asig, kfreq, kband [, iskip]
ares butterhp asig, kfreq [, iskip]
ares butterlp asig, kfreq [, iskip]
ares clfilt asig, kfreq, itype, inpol [, ikind] [, ipbr] [, isba] [, iskip]
ashifted doppler asource, ksourceposition, kmicposition [, isoundspeed, ifiltercutoff]
aout mode ain, kfreq, kQ [, iskip]
ares tone asig, khp [, iskip]
ares tonex asig, khp [, inumlayer] [, iskip]
```

### Modificateurs de Signal : Filtres Standard : Résonants.

```
ares areson asig, kcf, kbw [, iscl] [, iskip]
ares bqrez asig, xfco, xres [, imode] [, iskip]
ares lowpass2 asig, kcf, kq [, iskip]
ares lowres asig, kcutoff, kresonance [, iskip]
ares lowresx asig, kcutoff, kresonance [, inumlayer] [, iskip]
ares lpf18 asig, kfco, kres, kdist
asig moogladder ain, kcf, kres[, istor]
ares moogvcf asig, xfco, xres [, iscale, iskip]
ares moogvcf2 asig, xfco, xres [, iscale, iskip]
ares reson asig, kcf, kbw [, iscl] [, iskip]
ares resonr asig, kcf, kbw [, iscl] [, iskip]
ares resonx asig, kcf, kbw [, inumlayer] [, iscl] [, iskip]
ares resony asig, kbf, kbw, inum, ksep [, isepmode] [, iscl] [, iskip]
ares resonz asig, kcf, kbw [, iscl] [, iskip]
ares rezzy asig, xfco, xres [, imode, iskip]
ahp,alp,abp,abr statevar ain, kcf, kq [, iosamps, istor]
alow, ahigh, aband svfilter asig, kcf, kq [, iscl]
ares tbvcf asig, xfco, xres, kdist, kasym [, iskip]
ares vlowres asig, kfco, kres, iord, ksep
```

### Modificateurs de Signal : Filtres Standard : Contrôle.

```
kres aresonk ksig, kcf, kbw [, iscl] [, iskip]
kres atonek ksig, khp [, iskip]
```

```

kres lineto ksig, ktime
kres port ksig, ihtim [, isig]
kres portk ksig, khtim [, isig]
kres resonk ksig, kcf, kbw [, iscl] [, iskip]
kres resonxk ksig, kcf, kbw[, inumlayer, iscl, istor]
kres tlineto ksig, ktime, ktrig
kres tonek ksig, khp [, iskip]

```

### Modificateurs de Signal : Filtres Spécialisés.

```

ares dcblock ain [, igain]
ares dcblock2 ain [, iorder] [, iskip]
asig eqfil ain, kcf, kbw, kgain[, istor]
ares filter2 asig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
kres filter2 ksig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
asig fofilter ain, kcf, kris, kdec[, istor]
ar1, ar2 hilbert asig
ares nlfilt ain, ka, kb, kd, kC, kL
ares pareq asig, kc, kv, kq [, imodel] [, iskip]
ar rbjeq asig, kfco, klvl, kQ, kS[, imodel]
ares zfilter2 asig, kdamp, kfreq, iM, iN, ib0, ib1, ..., ibM, \
    ia1,ia2, ..., iaN

```

### Modificateurs de Signal : Guides d'Onde.

```

ares wguide1 asig, xfreq, kcutoff, kfeedback
ares wguide2 asig, xfreq1, xfreq2, kcutoff1, kcutoff2, \
    kfeedback1, kfeedback2

```

### Modificateurs de Signal : Distorsion Non-Linéaire.

```

aout chebyshevpoly ain, k0 [, k1 [, k2 [...]]]
aout pdclip ain, kWidth, kCenter [, ibipolar [, ifullscale]]
aout pdhalf ain, kShapeAmount [, ibipolar [, ifullscale]]
aout pdhalfy ain, kShapeAmount [, ibipolar [, ifullscale]]
aout powershape ain, kShapeAmount [, ifullscale]

```

### Modificateurs de Signal : Comparateurs et Accumulateurs.

```
amax max ain1 [, ain2] [, ain3] [, ain4] [...]
kmax max kin1 [, kin2] [, kin3] [, kin4] [...]

knumkout max_k asig, ktrig, itype

amax maxabs ain1 [, ain2] [, ain3] [, ain4] [...]
kmax maxabs kin1 [, kin2] [, kin3] [, kin4] [...]

maxabsaccum aAccumulator, aInput

maxaccum aAccumulator, aInput

amin min ain1 [, ain2] [, ain3] [, ain4] [...]
kmin min kin1 [, kin2] [, kin3] [, kin4] [...]

amin minabs ain1 [, ain2] [, ain3] [, ain4] [...]
kmin minabs kin1 [, kin2] [, kin3] [, kin4] [...]

minabsaccum aAccumulator, aInput

minaccum aAccumulator, aInput
```

### Contrôle d'Instrument : Contrôle d'Horloge.

```
clockoff inum

clockon inum
```

### Contrôle d'Instrument : Valeurs Conditionnelles.

```
(a == b ? v1 : v2)

(a >= b ? v1 : v2)

(a > b ? v1 : v2)

(a <= b ? v1 : v2)

(a < b ? v1 : v2)

(a != b ? v1 : v2)
```

### Contrôle d'Instrument : Contrôle de Durée.

```
ihold

turnoff

turnoff2 kinsno, kmode, krelease

turnon insnum [, itime]
```

### Contrôle d'Instrument : Appel d'Instrument.

```
event "scorechar", kinsnum, kdelay, kdur, [, kp4] [, kp5] [, ...]
event "scorechar", "insname", kdelay, kdur, [, kp4] [, kp5] [, ...]
```

```

event_i "scorechar", iinsnum, idelay, idur, [, ip4] [, ip5] [, ...]
event_i "scorechar", "insname", idelay, idur, [, ip4] [, ip5] [, ...]

mute insnum [, iswitch]
mute "insname" [, iswitch]

remove insnum

schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur \
[, ip4] [, ip5] [...]
schedkwhen ktrigger, kmintim, kmaxnum, "insname", kwhen, kdur \
[, ip4] [, ip5] [...]

schedkwhennamed ktrigger, kmintim, kmaxnum, "name", kwhen, kdur \
[, ip4] [, ip5] [...]

schedule insnum, iwhen, idur [, ip4] [, ip5] [...]
schedule "insname", iwhen, idur [, ip4] [, ip5] [...]

schedwhen ktrigger, kinsnum, kwhen, kdur [, ip4] [, ip5] [...]
schedwhen ktrigger, "insname", kwhen, kdur [, ip4] [, ip5] [...]

scoreline Sin, ktrig

scoreline_i Sin

```

### Contrôle d'Instrument : Contrôle Séquentiel d'un Programme.

```

cggoto condition, label

cigoto condition, label

ckgoto condition, label

cngoto condition, label

else

elseif xa R xb then

endif

goto label

if ia R ib igoto label
if ka R kb kgoto label
if xa R xb goto label
if xa R xb then

igoto label

kgoto label

loop_ge   indx, idecr, imin, label
loop_ge   kndx, kdecr, kmin, label

loop_gt   indx, idecr, imin, label
loop_gt   kndx, kdecr, kmin, label

loop_le   indx, incr, imax, label
loop_le   kndx, kncr, kmax, label

loop_lt   indx, incr, imax, label
loop_lt   kndx, kncr, kmax, label

tigoto label

timeout istrt, idur, label

```

### Contrôle d'Instrument : Contrôle de l'Exécution en Temps Réel.

```

ir active insnum
kres active kinsnum

cpuprc insnum, ipercent

exitnow

jacktransport icommand [, ilocation]

maxalloc insnum, icount

prealloc insnum, icount
prealloc "insname", icount

```

### Contrôle d'Instrument : Initialisation et Réinitialisation.

```

ares = xarg
ires = iarg
kres = karg
ires, ... = iarg, ...
kres, ... = karg, ...

ares init iarg
ires init iarg
kres init iarg
ares, ... init iarg, ...
ires, ... init iarg, ...
kres, ... init iarg, ...

insno nstrnum "name"

p(x)

pset icon1 [, icon2] [...]

reinit label

rigoto label

rireturn

ir tival

```

### Contrôle d'Instrument : Détection et Contrôle.

```

kres button knum

ktrig changed kvar1 [, kvar2,..., kvarN]

kres checkbox knum

kres control knum

ares follow asig, idt

ares follow2 asig, katt, krel

Svalue getcfig iopt

ktrig metro kfreq [, initphase]

ksig miditempo

```



## p5gconnect

```

kres p5gdata kcontrol

icount pcount

kres peak asig
kres peak ksig

ivalue pindex ipfieldIndex

koct, kamp pitch asig, iupdte, ilo, ihi, idbthresh [, ifrqs] [, iconf] \
    [, istrtr] [, iocts] [, iq] [, inptls] [, irolloff] [, iskip]

kcps, krms pitchamdf asig, imincps, imaxcps [, icps] [, imedi] \
    [, idowns] [, iexcps] [, irmsmedi]

kcps, kamp ptrack asig, ihopsize[,ipeaks]

rewindscore

kres rms asig [, ihp] [, iskip]

kres[, kkeydown] sensekey

ktrig_out seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times

ktrig_out seqtime2 ktrig_in, ktime_unit, kstart, kloop, kinitndx, kfn_times

setctrl inum, ival, itype

setscorepos ipos

splitrig ktrig, kndx, imaxtics, ifn, kout1 [,kout2,...,koutN]

ktemp tempest kin, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, \
    istartempo, ifn [, idisprd] [, itweek]

tempo ktempo, istartempo

kres tempoval

ktrig timedseq ktimpnt, ifn, kp1 [,kp2, kp3, ...,kpN]

kout trigger ksig, kthreshold, kmode

trigseq ktrig_in, kstart, kloop, kinitndx, kfn_values, kout1 [, kout2] [...]

kres wiiconnect [ittimeout, imaxnum]

kres wiidata kcontrol[, knum]

wiirange icontrol, iminimum, imaximum[, inum]

kres wiisend kcontrol, kvalue[, knum]

kx, ky xyin iprd, ixmin, ixmax, iymmin, iymax [, ixinit] [, iyinit]
```

## Contrôle d'Instrument : Piles.

```

xval1, [xval2, ... , xval31] pop
ival1, [ival2, ... , ival31] pop

fsig pop_f

push xval1, [xval2, ... , xval31]
push ival1, [ival2, ... , ival31]

push_f fsig
```

**stack** iStackSize

### Contrôle d'Instrument : Contrôle de sous-instrument.

```
al, [...] [, a8] subinstr instrnum [, p4] [, p5] [...]
al, [...] [, a8] subinstr "insname" [, p4] [, p5] [...]

subinstrinit instrnum [, p4] [, p5] [...]
subinstrinit "insname" [, p4] [, p5] [...]
```

### Contrôle d'Instrument : Lecture du Temps.

```
ir date

Sir dates [ itime]

ir readclock inum

ires rtclock
kres rtclock

kres timeinstk

kres timeinsts

ires timek
kres timek

ires times
kres times
```

### Contrôle des Tables de Fonction.

```
ftfree ifno, iwhen

gir ftgen ifn, itime, isize, igen, iarga [, iargb ] [...]

ifno ftgenonce ipldummy, ip2dummy, isize, igen, iarga, iargb, ...

ifno ftgentmp ip1, ip2dummy, isize, igen, iarga, iargb, ...

sndload Sfname[, ifmt[, ichns[, isr[, ibas[, iamp[, istr  \
    [, ilpmod[, ilps[, ilpe]]]]]]]]]
```

### Contrôle des Tables de Fonction : Requêtes sur une Table.

```
ftchnls(x) (arg de taux-i seulement)

ftlen(x) (arg de taux-i seulement)

ftlptim(x) (arg de taux-i seulement)

ftsrr(x) (arg de taux-i seulement)

nsamp(x) (arg de taux-i seulement)

ires tableng ifn
kres tableng kfn
```

```
kr tabsum ifn[[, kmin] [, kmax]]
```

```
tb0_init ifn
tb1_init ifn
tb2_init ifn
tb3_init ifn
tb4_init ifn
tb5_init ifn
tb6_init ifn
tb7_init ifn
tb8_init ifn
tb9_init ifn
tb10_init ifn
tb11_init ifn
tb12_init ifn
tb13_init ifn
tb14_init ifn
tb15_init ifn
iout = tb0(iIndex)
kout = tb0(kIndex)
iout = tb1(iIndex)
kout = tb1(kIndex)
iout = tb2(iIndex)
kout = tb2(kIndex)
iout = tb3(iIndex)
kout = tb3(kIndex)
iout = tb4(iIndex)
kout = tb4(kIndex)
iout = tb5(iIndex)
kout = tb5(kIndex)
iout = tb6(iIndex)
kout = tb6(kIndex)
iout = tb7(iIndex)
kout = tb7(kIndex)
iout = tb8(iIndex)
kout = tb8(kIndex)
iout = tb9(iIndex)
kout = tb9(kIndex)
iout = tb10(iIndex)
kout = tb10(kIndex)
iout = tb11(iIndex)
kout = tb11(kIndex)
iout = tb12(iIndex)
kout = tb12(kIndex)
iout = tb13(iIndex)
kout = tb13(kIndex)
iout = tb14(iIndex)
kout = tb14(kIndex)
iout = tb15(iIndex)
kout = tb15(kIndex)
```

### Contrôle des Tables de Fonction : Sélection Dynamique.

```
ares tableikt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
kres tableikt kndx, kfn [, ixmode] [, ixoff] [, iwrap]

ares tablekt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
kres tablekt kndx, kfn [, ixmode] [, ixoff] [, iwrap]

ares tablexkt xndx, kfn, kwarp, iwsiz [, ixmode] [, ixoff] [, iwrap]
```

### Contrôle des Tables de Fonction : Opérations de Lecture/Ecriture.

```
ftload "filename", iflag, ifn1 [, ifn2] [...]
ftloadk "filename", ktrig, iflag, ifn1 [, ifn2] [...]
ftsave "filename", iflag, ifn1 [, ifn2] [...]
ftsavek "filename", ktrig, iflag, ifn1 [, ifn2] [...]
```

```

tablecopy kdft, ksft

tablegpw kfn

tableicopy idft, isft

tableigpw ifn

tableimix idft, idoff, ilen, islft, isloff, islg, is2ft, is2off, is2g

tableiw isig, indx, ifn [, ixmode] [, ixoff] [, iwgmde]

tablemix kdft, kdoff, klen, kslft, ksoff, kslg, ks2ft, ks2off, ks2g

ares tablera kfn, kstart, koff

tablew asig, andx, ifn [, ixmode] [, ixoff] [, iwgmde]
tablew isig, indx, ifn [, ixmode] [, ixoff] [, iwgmde]
tablew ksig, kndx, ifn [, ixmode] [, ixoff] [, iwgmde]

kstart tablewa kfn, asig, koff

tablewkt asig, andx, kfn [, ixmode] [, ixoff] [, iwgmde]
tablewkt ksig, kndx, kfn [, ixmode] [, ixoff] [, iwgmde]

kout tabmorph kindex, kweightpoint, ktabnum1, ktabnum2, \
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

aout tabmorpha aindex, aweightpoint, atabnum1, atabnum2, \
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

aout tabmorphak aindex, kweightpoint, ktabnum1, ktabnum2, \
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

kout tabmorphi kindex, kweightpoint, ktabnum1, ktabnum2, \
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

tabplay ktrig, knumtics, kfn, kout1 [,kout2,..., koutN]

tabrec ktrig_start, ktrig_stop, knumtics, kfn, kin1 [,kin2,...,kinN]

```

## FLTK : Conteneurs.

```

FLgroup "label", iwidth, iheight, ix, iy [, iborder] [, image]

FLgroupEnd

FLpack iwidth, iheight, ix, iy, itype, ispace, iborder

FLpackEnd

FLpanel "label", iwidth, iheight [, ix] [, iy] [, iborder] [, ikbdcapture] [, iclose]

FLpanelEnd

FLscroll iwidth, iheight [, ix] [, iy]

FLscrollEnd

FLtabs iwidth, iheight, ix, iy

FLtabsEnd

```

## FLTK : Valuateurs.

```
kout, ihandle FLcount "label", imin, imax, istep1, istep2, itype, \
    iwidth, iheight, ix, iy, iopcode [, kp1] [, kp2] [, kp3] [...] [, kpN]

koutx, kouty, ihandlex, ihandley FLjoy "label", iminx, imaxx, iminy, \
    imaxy, iexp, iexpy, idisp, idispy, iwidth, iheight, ix, iy

kout, ihandle FLknob "label", imin, imax, iexp, itype, idisp, iwidth, \
    ix, iy [, icursorsize]

kout, ihandle FLroller "label", imin, imax, istep, iexp, itype, idisp, \
    iwidth, iheight, ix, iy

kout, ihandle FLslider "label", imin, imax, iexp, itype, idisp, iwidth, \
    iheight, ix, iy

kout, ihandle FLtext "label", imin, imax, istep, itype, iwidth, \
    iheight, ix, iy
```

### FLTK : Autres.

```
ihandle FLbox "label", itype, ifont, isize, iwidth, iheight, ix, iy [, image]

kout, ihandle FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy, \
    iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [...] [, kpN]

kout, ihandle FLbutton "label", ion, ioff, itype, iwidth, iheight, ix, \
    iy, iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [...] [, kpN]

ihandle FLcloseButton "label", iwidth, iheight, ix, iy

ihandle FLexecButton "command", iwidth, iheight, ix, iy

inumsnap FLgetsnap index [, igroup]

ihandle FLhvsBox inumlinesX, inumlinesY, iwidth, iheight, ix, iy [, image]

FLhvsBox kx, ky, ihandle

kascii FLkeyIn [ifn]

FLloadsnap "filename" [, igroup]

kx, ky, kb1, kb2, kb3 FLmouse [imodel]

FLprintk itime, kval, idisp

FLprintk2 kval, idisp

FLrun

FLsavesnap "filename" [, igroup]

inumsnap, inumval FLsetsnap index [, ifn, igroup]

FLsetSnapGroup igroup

FLsetVal ktrig, kvalue, ihandle

FLsetVal_i ivalue, ihandle

FLslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
    [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]

FLslidBnk2 "names", inumsliders, ioutable, iconfigtable [, iwidth, iheight, ix, iy, is-
    tart_index]

FLslidBnk2Set ihandle, ifn [, istartIndex, istartSlid, inumSlid]

FLslidBnk2Setk ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

```

ihandle FLslidBnkGetHandle

FLslidBnkSet ihandle, ifn [, istartIndex, istartSlid, inumSlid]

FLslidBnkSetk ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]

FLupdate

ihandle FLvalue "label", iwidth, iheight, ix, iy

FLvkeybd "keyboard.map", iwidth, iheight, ix, iy

FLvslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
    [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]

FLvslidBnk2 "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, is-
    tart_index]

koutx, kouty, kinside FLxyin ioutx_min, ioutx_max, iouty_min, iouty_max, \
    iwindx_min, iwindx_max, iwindy_min, iwindy_max [, iexpv, iexpv, ioutx, iouty]

vphaseseg kphase, ioutab, ielems, itab1,idist1,itab2 \
    [,idist2,itab3, ... ,idistN-1,itabN]

```

### FLTK : Apparence.

```

FLcolor ired, igreen, iblue [, ired2, igreen2, iblue2]

FLcolor2 ired, igreen, iblue

FLhide ihandle

FLlabel isize, ifont, ialign, ired, igreen, iblue

FLsetAlign ialign, ihandle

FLsetBox itype, ihandle

FLsetColor ired, igreen, iblue, ihandle

FLsetColor2 ired, igreen, iblue, ihandle

FLsetFont ifont, ihandle

FLsetPosition ix, iy, ihandle

FLsetSize iwidth, iheight, ihandle

FLsetText "itext", ihandle

FLsetTextColor ired, iblue, igreen, ihandle

FLsetTextSize isize, ihandle

FLsetTextType itype, ihandle

FLshow ihandle

```

### Opérations Mathématiques : Compérateurs et Accumulateurs.

```

a + b (no rate restriction)

a / b (no rate restriction)

a % b (no rate restriction)

```

`a * b` (no rate restriction)  
`a && b` (ET logique ; pas de taux audio)  
`a & b` (ET binaire)  
`~ a` (NON binaire)  
`a | b` (bitwise OR)  
`a << b` (bitshift left)  
`a >> b` (bitshift left)  
`a # b` (NON-EQUIVALENCE binaire)  
`a || b` (logical OR; not audio-rate)  
`a ^ b` (b not audio-rate)  
`a # b` (no rate restriction)

### Opérations Mathématiques : Opérations Arithmétiques et Logiques.

`clear` avar1 [, avar2] [, avar3] [...]  
`vincr` accum, aincr

### Opérations Mathématiques : Fonctions Mathématiques.

`abs`(x) (pas de restriction de taux)  
`ceil`(x) (argument au taux d'initialisation, de contrôle ou audio)  
`exp`(x) (pas de restriction de taux)  
`floor`(x) (argument au taux d'initialisation, de contrôle ou audio)  
`frac`(x) (arguments de taux-i ou de taux-k ; fonctionne aussi au taux-a dans Csound5)  
`int`(x) (taux-i ou taux-k ; fonctionne aussi au taux-a dans Csound5)  
`log`(x) (pas de restriction de taux)  
`log10`(x) (pas de restriction de taux)  
`logbtwo`(x) (argument au taux d'initialisation ou de contrôle seulement)  
`powoftwo`(x) (argument au taux d'initialisation ou de contrôle seulement)  
`round`(x) (des arguments de taux-i, -k ou -a sont permis)  
`sqrt`(x) (pas de restriction de taux)

### Opérations Mathématiques : Fonctions Trigonométriques.

`cos`(x) (pas de restriction de taux)  
`cosh`(x) (pas de restriction de taux)

`cosinv`(x) (pas de restriction de taux)  
`sin`(x) (pas de restriction de taux)  
`sinh`(x) (pas de restriction de taux)  
`sininv`(x) (pas de restriction de taux)  
`tan`(x) (pas de restriction de taux)  
`tanh`(x) (pas de restriction de taux)  
`taninv`(x) (pas de restriction de taux)

### Opérations Mathématiques : Fonctions d'Amplitude.

`ampdb`(x) (pas de restriction de taux)  
`ampdbfs`(x) (pas de restriction de taux)  
`db`(x)  
`dbamp`(x) (arguments de taux-i ou -k seulement)  
`dbfsamp`(x) (arguments de taux-i ou -k seulement)

### Opérations Mathématiques : Fonctions aléatoires.

`birnd`(x) (taux-i ou -k seulement)  
`rnd`(x) (taux-i ou -k seulement)

### Opérations Mathématiques : Opcodes Equivalents à des Fonctions.

ares `divz` xa, xb, ksubst  
ires `divz` ia, ib, isubst  
kres `divz` ka, kb, ksubst

ares `mac` asig1, ksig1 [, asig2] [, ksig2] [, asig3] [, ksig3] [...]

ares `maca` asig1 , asig2 [, asig3] [, asig4] [, asig5] [...]

aout `polynomial` ain, k0 [, k1 [, k2 [...]]]

ares `pow` aarg, kpow [, inorm]  
ires `pow` iarg, ipow [, inorm]  
kres `pow` karg, kpow [, inorm]

ares `product` asig1, asig2 [, asig3] [...]

ares `sum` asig1 [, asig2] [, asig3] [...]

ares `taninv2` ay, ax  
ires `taninv2` iy, ix  
kres `taninv2` ky, kx

### Conversion des Hauteurs : Fonctions.



**cent**(x)

**cpsmidinn** (MidiNoteNumber) (arguments de taux-i ou -k seulement)

**cpsoct** (oct) (pas de restriction de taux)

**cpspch** (pch) (arguments de taux-i ou -k seulement)

**octave**(x)

**octcps** (cps) (arguments de taux-i ou -k seulement)

**octmidinn** (MidiNoteNumber) (arguments de taux-i ou -k seulement)

**octpch** (pch) (arguments de taux-i ou -k seulement)

**pchmidinn** (MidiNoteNumber) (arguments de taux-i ou -k seulement)

**pchoct** (oct) (arguments de taux-i ou -k seulement)

**semitone**(x)

### Conversion des Hauteurs : Opcodes de Hauteurs.

icps **cps2pch** ipch, iequal

kcps **cpstun** ktrig, kindex, kfn

icps **cpstuni** index, ifn

icps **cpsxpch** ipch, iequal, irepeat, ibase

### MIDI en Temps-Réel : Entrée.

kaft **aftouch** [imin] [, imax]

ival **chanctrl** ichnl, ictlno [, ilow] [, ihigh]  
kval **chanctrl** ichnl, ictlno [, ilow] [, ihigh]

idest **ctrl14** ichan, ictlno1, ictlno2, imin, imax [, ifn]  
kdest **ctrl14** ichan, ictlno1, ictlno2, kmin, kmax [, ifn]

idest **ctrl21** ichan, ictlno1, ictlno2, ictlno3, imin, imax [, ifn]  
kdest **ctrl21** ichan, ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]

idest **ctrl7** ichan, ictlno, imin, imax [, ifn]  
kdest **ctrl7** ichan, ictlno, kmin, kmax [, ifn]  
adest **ctrl7** ichan, ictlno, kmin, kmax [, ifn] [, icutoff]

**ctrlinit** ichnl, ictlno1, ival1 [, ictlno2] [, ival2] [, ictlno3] \  
[, ival3] [,...ival32]

**initc14** ichan, ictlno1, ictlno2, ivalue

**initc21** ichan, ictlno1, ictlno2, ictlno3, ivalue

**initc7** ichan, ictlno, ivalue

**massign** ichnl, insnum[, ireset]  
**massign** ichnl, "insname"[, ireset]

idest **midic14** ictlno1, ictlno2, imin, imax [, ifn]  
kdest **midic14** ictlno1, ictlno2, kmin, kmax [, ifn]

idest **midic21** ictlno1, ictlno2, ictlno3, imin, imax [, ifn]  
kdest **midic21** ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]

```

idest midic7 ictlno, imin, imax [, ifn]
kdest midic7 ictlno, kmin, kmax [, ifn]

ival midictrl inum [, imin] [, imax]
kval midictrl inum [, imin] [, imax]

ival notnum

ibend pchbend [imin] [, imax]
kbend pchbend [imin] [, imax]

pgmassign ipgm, inst[, ichn]
pgmassign ipgm, "insname"[, ichn]

ires polyaft inote [, ilow] [, ihigh]
kres polyaft inote [, ilow] [, ihigh]

ival veloc [ilow] [, ihigh]

```

### MIDI en Temps-Réel : Sortie.

```

nrpn kchan, kparamnum, kparamvalue

outiat ichn, ivalue, imin, imax

outic ichn, inum, ivalue, imin, imax

outic14 ichn, imsb, ilsb, ivalue, imin, imax

outipat ichn, inotenum, ivalue, imin, imax

outipb ichn, ivalue, imin, imax

outipc ichn, iprog, imin, imax

outkat kchn, kvalue, kmin, kmax

outkc kchn, knum, kvalue, kmin, kmax

outkc14 kchn, kmsb, klsb, kvalue, kmin, kmax

outkpat kchn, knotenum, kvalue, kmin, kmax

outkpb kchn, kvalue, kmin, kmax

outkpc kchn, kprog, kmin, kmax

```

### MIDI en Temps-Réel : Convertisseurs.

```

iamp ampmidi iscal [, ifn]

icps cpsmidi

icps cpsmidib [irange]
kcps cpsmidib [irange]

icps cpstmid ifn

ioct octmidi

ioct octmidib [irange]
koct octmidib [irange]

ipch pchmidi

```

ipch **pchmidib** [irange]  
kpch **pchmidib** [irange]

### **MIDI en Temps-Réel : E/S Génériques.**

kstatus, kchan, kdata1, kdata2 **midiin**  
**midiout** kstatus, kchan, kdata1, kdata2

### **MIDI en Temps-Réel : Extension d'Evènements.**

kflag **release**  
**xtratim** iextradur

### **MIDI en Temps-Réel : Sortie de Note.**

**midion** kchn, knum, kvel  
**midion2** kchn, knum, kvel, ktrig  
**moscil** kchn, knum, kvel, kdur, kpause  
**noteoff** ichn, inum, ivel  
**noteon** ichn, inum, ivel  
**noteondur** ichn, inum, ivel, idur  
**noteondur2** ichn, inum, ivel, idur

### **MIDI en Temps-Réel : Interopérabilité MIDI/Partition.**

**midichannelaftertouch** xchannelaftertouch [, ilow] [, ihigh]  
ichn **midichn**  
**midicontrolchange** xcontroller, xcontrollervalue [, ilow] [, ihigh]  
**mididefault** xdefault, xvalue  
**midinoteoff** xkey, xvelocity  
**midinoteoncps** xcps, xvelocity  
**midinoteonkey** xkey, xvelocity  
**midinoteonoct** xoct, xvelocity  
**midinoteonpch** xpch, xvelocity  
**midipitchbend** xpitchbend [, ilow] [, ihigh]  
**midipolyaftertouch** xpolyaftertouch, xcontrollervalue [, ilow] [, ihigh]  
**midiprogramchange** xprogram

## MIDI en Temps-Réel : System Realtime.

**mclock** ifreq

**mrtmsg** imsgtype

## MIDI en Temps-Réel : Banques de Réglettes.

```

i1,...,i16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
    initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16,
ifn16
k1,...,k16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
    initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16,
ifn16

i1,...,i32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
    initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32,
ifn32
k1,...,k32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
    initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32,
ifn32

i1,...,i16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum16, imin16, imax16, init16, ifn16
k1,...,k16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum16, imin16, imax16, init16, ifn16

k1,...,k16 slider16f ichan, ictlnum1, imin1, imax1, init1, ifn1, \
    icutoff1,..., ictlnum16, imin16, imax16, init16, ifn16, icutoff16

kflag slider16table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, .... , ictlnum16, imin16, imax16, init16, ifn16

kflag slider16tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum16, imin16, imax16, init16, ifn16, icutoff16

i1,...,i32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum32, imin32, imax32, init32, ifn32
k1,...,k32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum32, imin32, imax32, init32, ifn32

k1,...,k32 slider32f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \
    ..., ictlnum32, imin32, imax32, init32, ifn32, icutoff32

kflag slider32table ichan, ioutTable, ioffset, ictlnum1, imin1, \
    imax1, init1, ifn1, .... , ictlnum32, imin32, imax32, init32, ifn32

kflag slider32tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum32, imin32, imax32, init32, ifn32, icutoff32

i1,...,i64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum64, imin64, imax64, init64, ifn64
k1,...,k64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum64, imin64, imax64, init64, ifn64

k1,...,k64 slider64f ichan, ictlnum1, imin1, imax1, init1, ifn1, \
    icutoff1,..., ictlnum64, imin64, imax64, init64, ifn64, icutoff64

kflag slider64table ichan, ioutTable, ioffset, ictlnum1, imin1, \
    imax1, init1, ifn1, .... , ictlnum64, imin64, imax64, init64, ifn64

kflag slider64tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum64, imin64, imax64, init64, ifn64, icutoff64

i1,...,i8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum8, imin8, imax8, init8, ifn8
k1,...,k8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum8, imin8, imax8, init8, ifn8

```

```

k1,...,k8 slider8f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \
    ..., ictlnum8, imin8, imax8, init8, ifn8, icutoff8

kflag slider8table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1,..., ictlnum8, imin8, imax8, init8, ifn8

kflag slider8tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum8, imin8, imax8, init8, ifn8, icutoff8

k1, k2, ..., k16 sliderKawai imin1, imax1, init1, ifn1, \
    imin2, imax2, init2, ifn2, ..., imin16, imax16, init16, ifn16

```

## Graphe de Fluence.

```

alwayson Tinstrument [p4, ..., pn]

connect Tsource1, Soutlet1, Tsink1, Sinlet1

asignal inleta Sname

fsignal inletf Sname

ksignal inletk Sname

outleta Sname, asignal

outletf Sname, fsignal

outletk Sname, ksignal

```

## Traitement Spectral : STFT.

```

tableseeg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]

ares pvadd ktmpnt, kfmmod, ifilcod, ifn, ibins [, ibinoffset] \
    [, ibinincr] [, iextractmode] [, ifreqlim] [, igatefn]

pvbufread ktmpnt, ifile

ares pvcross ktmpnt, kfmmod, ifile, kampscale1, kampscale2 [, ispecwp]

ares pvinterp ktmpnt, kfmmod, ifile, kfregscale1, kfregscale2, \
    kampscale1, kampscale2, kfreginterp, kampinterp

ares pvoc ktmpnt, kfmmod, ifilcod [, ispecwp] [, iextractmode] \
    [, ifreqlim] [, igatefn]

kfreg, kamp pvread ktmpnt, ifile, ibin

tableseeg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]

tablexseeg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]

ares pvvoc ktmpnt, kfmmod, ifile [, ispecwp] [, ifn]

```

## Traitement Spectral : LPC.

```

ares lpfreson asig, kfrqratio

lpinterp islot1, islot2, kmix

```

```
krmsr, krms0, kerr, kcps lpread ktimpnt, ifilcod [, inpoles] [, ifrmrate]
ares lpreson asig
lpslot islot
```

### Traitement Spectral : Non-Standard.

```
wsig specaddm wsig1, wsig2 [, imul2]
wsig specdiff wsigin
specdisp wsig, iprd [, iwtflg]
wsig specfilt wsigin, ifhtim
wsig spechist wsigin
koct, kamp specptrk wsig, kvar, ilo, ihi, istr, idbthresh, inptls, \
    irolloff [, iodd] [, iconfs] [, interp] [, ifprd] [, iwtflg]
wsig specscal wsigin, ifscale, ifthresh
ksum specsum wsig [, interp]
wsig spectrum xsig, iprd, iocts, ifrq [, iq] [, ihann] [, idbout] \
    [, idsprd] [, idsinrs]
```

### Traitement Spectral : Streaming.

```
fsg binit fin, isize
ftrks partials ffr, fphs, kthresh, kminpts, kmaxgap, imaxtracks
ares pvsadsyn fsrc, inoscs, kfmod [, ibinoffset] [, ibinincr] [, iinit]
fsg pvsanal ain, ifftsize, ioverlap, iwinsize, iwintype [, iformat] [, iinit]
fsg pvsarp fsigin, kbin, kdepth, kgain
fsg pvsbandp fsigin, xlowcut,
    xlowfull, xhighfull, xhighcut[, ktype]
fsg pvsbandr fsigin, xlowcut,
    xlowfull, xhighfull, xhighcut[, ktype]
kamp, kfr pvsbin fsg, kbin
fsg pvsblur fsigin, kblurtime, imaxdel
ihandle, ktime pvsbuffer fsg, ilen
fsg pvsbufread ktime, khandle[, ilo, ihi]
fsg pvscale fsigin, kscal[, ikeepform, igain]]
kcent pvscent fsg
fsg pvsccross fsrc, fdest, kamp1, kamp2
fsg pvsdemix fleft, fright, kpos, kwidth, ipoints
fsg pvsdiskin Sfname, ktscal, kgain[, ioffset, ichan]
pvsdisp fsg[, ibins, iwtflg]
```

```

fsig pvsfilter fsigin, fsigfil, kdepth[, igain]

fsig pvsfread ktimpt, ifn [, ichan]

fsig pvsfreeze fsigin, kfreeza, kfreezf

pvsftr fsrc, ifna [, ifnf]

kflag pvsftw fsrc, ifna [, ifnf]

pvsfwrite fsig, ifile

fsig pvsshift fsigin, kshift, klowest[, ikeepform, igain]

fpr,fphs pvsifd ain, ifftsize, ihopsize, iwintype[,iscal]

fsig pvsin kchan[,isize,iolap,iwinsize,iwintype,iformat]

ioverlap, inumbins, iwinsize, iformat pvsinfo fsrc

fsig pvsinit isize[,iolap,iwinsize,iwintype, iformat]

fsig pvsmaska fsrc, ifn, kdepth

fsig pvsmix fsigin1, fsigin2

fsig pvssmooth fsigin, kacf, kfcf

fsig pvsmorph fsig1, fsig2, kampint, kfrqint

fsig pvsosc kamp, kfreq, ktype, isize [,ioverlap] [, iwinsize] [, iwintype] [, iformat]

pvsout fsig, kchan

kfr, kamp pvspitch fsig, kthresh

fsig pvsstencil fsigin, kgain, klevel, iftable

fsig pvsvoc famp, fexc, kdepth, kgain

ares pvsynth fsrc, [iinit]

asig resyn fin, kscal, kpitch, kmaxtracks, ifn

asig sinsyn fin, kscal, kmaxtracks, ifn

asig tradsyn fin, kscal, kpitch, kmaxtracks, ifn

fsig trcross fin1, fin2, ksearch,kdepth[,kmode]

fsig trfilter fin, kamnt, ifn

fsig, kfr,kamp trhighest fin1, kscal

fsig, kfr,kamp trlowest fin1, kscal

fsig trmix fin1, fin2

fsig trscale fin, kpitch[, kgain]

fsig trshift fin, kpshift[, kgain]

fsiglow, fsighi trsplrit fin, ksplit[, kgainlow, kgainhigh]

```

### Traitement Spectral : ATS.

```

ar ATSadd ktimepnt, kfmmod, iatsfile, ifn, ipartial[, ipartialoffset, \
    ipartialincr, igatefn]

```

```

ar ATSaddnz ktimepnt, iatsfile, ibands[, ibandoffset, ibandincr]

ATSbufread ktimepnt, kfmmod, iatsfile, ipartials[, ipartialoffset, \
    ipartialincr]

ar ATScross ktimepnt, kfmmod, iatsfile, ifn, kmylev, kbuflev, ipartials \
    [, ipartialoffset, ipartialincr]

idata ATSinfo iatsfile, ilocation

kamp ATSinterpread kfreq

kfrq, kamp ATSpartialtap ipartialnum

kfreq, kamp ATSread ktimepnt, iatsfile, ipartial

kenenergy ATSreadnz ktimepnt, iatsfile, iband

ar ATSSinnoi ktimepnt, ksinlev, knzlev, kfmmod, iatsfile, ipartials \
    [, ipartialoffset, ipartialincr]

```

### Traitement Spectral : Loris.

```

lorismorph isrcidx, itgtidx, istoreidx, kfreqmorphenv, kampmorphenv, kbwmorphenv

ar lorisplay ireadidx, kfreqenv, kampenv, kbwenv

lorisread ktimepnt, ifilcod, istoreidx, kfreqenv, kampenv, kbwenv[, ifadetime]

```

### Chaînes : Définition.

```

Sdst strget indx

strset iarg, istring

```

### Chaînes : Manipulation.

```

puts Sstr, ktrig[, inonl]

Sdst sprintf Sfmt, xarg1[, xarg2[, ... ]]
Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]

Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]

Sdst strcat Ssrc1, Ssrc2

Sdst strcatk Ssrc1, Ssrc2

ires strcmp S1, S2

kres strcmpk S1, S2

Sdst strcpy Ssrc
Sdst = Ssrc

Sdst strcpyk Ssrc

ipos strindex S1, S2

kpos strindexk S1, S2

```



```
ilen strlen Sstr
klen strlenk Sstr
ipos strrindex S1, S2
kpos strrindexk S1, S2
Sdst strsub Ssrc[, istart[, iend]]
Sdst strsubk Ssrc, kstart, kend
```

### Chaînes : Conversion.

```
ichr strchar Sstr[, ipos]
kchr strchark Sstr[, kpos]
Sdst strlower Ssrc
Sdst strlowerk Ssrc

ir strtod Sstr
ir strtod indx

kr strtodk Sstr
kr strtodk kndx

ir strtol Sstr
ir strtol indx

kr strtolk Sstr
kr strtolk kndx

Sdst strupper Ssrc
Sdst strupperk Ssrc
```

### Vectoriel : Tableaux.

```
vtaba andx, ifn, aout1 [, aout2, aout3, .... , aoutN ]
vtabi indx, ifn, iout1 [, iout2, iout3, .... , ioutN ]
vtabk kndx, ifn, kout1 [, kout2, kout3, .... , koutN ]
vtablelk kfn,kout1 [, kout2, kout3, .... , koutN ]
vtablea andx, kfn, kinterp, ixmode, aout1 [, aout2, aout3, .... , aoutN ]
vtablei indx, ifn, interp, ixmode, iout1 [, iout2, iout3, .... , ioutN ]
vtablek kndx, kfn, kinterp, ixmode, kout1 [, kout2, kout3, .... , koutN ]
vtablewa andx, kfn, ixmode, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]
vtablewi indx, ifn, ixmode, inarg1 [, inarg2, inarg3 , .... , inargN ]
vtablewk kndx, kfn, ixmode, kinarg1 [, kinarg2, kinarg3 , .... , kinargN ]
vtabwa andx, ifn, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]
vtabwi indx, ifn, inarg1 [, inarg2, inarg3 , .... , inargN ]
vtabwk kndx, ifn, kinarg1 [, kinarg2, kinarg3 , .... , kinargN ]
```

### Vectoriel : Opérations Scalaires.

```

vadd ifn, kval, kelements [, kdstoffset] [, kverbose]
vadd_i ifn, ival, ielements [, idstoffset]
vexp ifn, kval, kelements [, kdstoffset] [, kverbose]
vexp_i ifn, ival, ielements[, idstoffset]
vmult ifn, kval, kelements [, kdstoffset] [, kverbose]
vmult_i ifn, ival, ielements [, idstoffset]
vpow ifn, kval, kelements [, kdstoffset] [, kverbose]
vpow_i ifn, ival, ielements [, idstoffset]

```

### Vectoriel : Opérations Vectorielles.

```

vaddv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vaddv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
vcopy ifn, ifn2, kelements [, kdstoffset] [, ksrcoffset] [, kverbose]
vcopy_i ifn, ifn2, ielements [,idstoffset, isrcoffset]
vdivv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vdivv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
vexpv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vexpv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
vmap ifn1, ifn2, ielements [,idstoffset, isrcoffset]
vmultv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vmultv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
vpowv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vpowv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
vsubv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vsubv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

```

### Vectoriel : Enveloppes.

```

vexpseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]
vlinseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]

```

### Vectoriel : Limitation et Enroulement.

```
vlimit ifn, kmin, kmax, ielements
vmirror ifn, kmin, kmax, ielements
vwrap ifn, kmin, kmax, ielements
```

### Vectoriel : Chemins de Retard.

```
kout vdelayk ksig, kdel, imaxdel [, iskip, imodel]
vecdelay ifn, ifnIn, ifnDel, ielements, imaxdel [, iskip]
vport ifn, khtime, ielements [, ifnInit]
```

### Vectoriel : Aléatoire.

```
vrandh ifn, krange, kcps, ielements [, idstoffset] [, iseed]
        [, isize] [, ioffset]
vrandi ifn, krange, kcps, ielements [, idstoffset] [, iseed]
        [, isize] [, ioffset]
```

### Vectoriel : Automates Cellulaires.

```
vcella ktrig, kreinit, ioutFunc, initStateFunc, \
        iRuleFunc, ielements, irulelen [, iradius]
```

### Système de Patch Zak.

```
zacl kfirst, klast
zakinit isizea, isizek
ares zamod asig, kzamod
ares zar kndx
ares zarg kndx, kgain
zaw asig, kndx
zawm asig, kndx [, imix]
ir zir indx
ziw isig, indx
ziwm isig, indx [, imix]
zkcl kfirst, klast
kres zkmod ksig, kzkmod
kres zkr kndx
zkw ksig, kndx
```

`zkwm` ksig, kndx [, imix]

### Accueil de Plugin : DSSI et LADSPA.

`dssiactivate` ihandle, ktoggle

aout1 [, aout2, aout3, aout4] `dssiaudio` ihandle, ain1 [,ain2, ain3, ain4]

`dssictls` ihandle, iport, kvalue, ktrigger

ihandle `dssiinit` ilibraryname, ipluginindex [, iverbose]

`dssilist`

### Accueil de Plugin : VST.

aout1,aout2 `vstaudio` instance, [ain1, ain2]  
aout1,aout2 `vstaudiog` instance, [ain1, ain2]

`vstbankload` instance, ipath

`vstedit` instance

`vstinfo` instance

instance `vstinit` ilibrarypath [,iverbose]

`vstmidiout` instance, kstatus, kchan, kdata1, kdata2

`vstnote` instance, kchan, knote, kveloc, kdur

`vstparamset` instance, kparam, kvalue  
kvalue `vstparamget` instance, kparam

`vstprogset` instance, kprogram

### OSC.

ihandle `OSCinit` iport

kans `OSClisten` ihandle, idest, itype [, xdata1, xdata2, ...]

`OSCsend` kwhen, ihost, iport, idestination, itype [, kdata1, kdata2, ...]

### Réseau.

`remoteport` iportnum

asig `sockrecv` iport, ilength  
asigl, asigr `sockrecvs` iport, ilength  
asig `strecv` Sipaddr, iport

`socksend` asig, Sipaddr, iport, ilength  
`socksends` asigl, asigr, Sipaddr, iport,  
ilength  
`stsend` asig, Sipaddr, iport

## Opcodes pour le Traitement à Distance.

```

insglobal isource, instrnum [,instrnum...]

insremot idestination, isource, instrnum [,instrnum...]

midglobal isource, instrnum [,instrnum...]

midremot idestination, isource, instrnum [,instrnum...]

```

## Opcodes Mixer.

### MixerClear

```

kgain MixerGetLevel isend, ibuss

asignal MixerReceive ibuss, ichannel

MixerSend asignal, isend, ibuss, ichannel

MixerSetLevel isend, ibuss, kgain

MixerSetLevel_i isend, ibuss, igain

```

## Opcodes Python.

```

pyassign "variable", kvalue
pyassigni "variable", ivalue
pylassign "variable", kvalue
pylassigni "variable", ivalue
pyassignt ktrigger, "variable", kvalue
pylassignt ktrigger, "variable", kvalue

kresult
kresult1, kresult2
kr1, kr2, kr3
kr1, kr2, kr3, kr4
kr1, kr2, kr3, kr4, kr5
kr1, kr2, kr3, kr4, kr5, kr6
kr1, kr2, kr3, kr4, kr5, kr6, kr7
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8

kresult
kresult1, kresult2
kr1, kr2, kr3
kr1, kr2, kr3, kr4
kr1, kr2, kr3, kr4, kr5
kr1, kr2, kr3, kr4, kr5, kr6
kr1, kr2, kr3, kr4, kr5, kr6, kr7
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8

iresult
iresult1, ireresult2
ir1, ir2, ir3
ir1, ir2, ir3, ir4
ir1, ir2, ir3, ir4, ir5
ir1, ir2, ir3, ir4, ir5, ir6
ir1, ir2, ir3, ir4, ir5, ir6, ir7
ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8
pycalln "callable", nresults, kresult1, ..., kresultn, karg1, ...
pycallni "callable", nresults, ireresult1, ..., ireresultn, iarg1, ...

kresult
kresult1, kresult2
pycall "callable", karg1, ...
pycall11 "callable", karg1, ...
pycall12 "callable", karg1, ...
pycall13 "callable", karg1, ...
pycall14 "callable", karg1, ...
pycall15 "callable", karg1, ...
pycall16 "callable", karg1, ...
pycall17 "callable", karg1, ...
pycall18 "callable", karg1, ...
pycall1t ktrigger, "callable", karg1, ...
pycall11t ktrigger, "callable", karg1, ...
pycall12t ktrigger, "callable", karg1, ...
pycall13t ktrigger, "callable", karg1, ...
pycall14t ktrigger, "callable", karg1, ...
pycall15t ktrigger, "callable", karg1, ...
pycall16t ktrigger, "callable", karg1, ...
pycall17t ktrigger, "callable", karg1, ...
pycall18t ktrigger, "callable", karg1, ...
pycalli "callable", iarg1, ...
pycall11i "callable", iarg1, ...
pycall12i "callable", iarg1, ...
pycall13i "callable", iarg1, ...
pycall14i "callable", iarg1, ...
pycall15i "callable", iarg1, ...
pycall16i "callable", iarg1, ...
pycall17i "callable", iarg1, ...
pycall18i "callable", iarg1, ...
pylcall "callable", karg1, ...
pylcall11 "callable", karg1, ...
pylcall12 "callable", karg1, ...

```

```

kr1, kr2, kr3      pylcall13 "callable", karg1, ...
kr1, kr2, kr3, kr4 pylcall14 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5 pylcall15 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6 pylcall16 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7 pylcall17 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall18 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall1t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall11t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall12t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall13t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall14t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall15t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall16t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall17t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall18t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall1i "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall11i "callable", iarg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall12i "callable", iarg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall13i "callable", iarg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall14i "callable", iarg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall15i "callable", iarg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall16i "callable", iarg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall17i "callable", iarg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8 pylcall18i "callable", iarg1, ...
pylcalln "callable", nresults, kresult1, ..., kresultn, karg1, ...
pylcallni "callable", nresults, irestult1, ..., irestultn, iarg1, ...

kresult pyeval "expression"
irestult pyevali "expression"
kresult pyleval "expression"
irestult pylevali "expression"
kresult pyevalt ktrigger, "expression"
kresult pylevalt ktrigger, "expression"

pyexec "filename"
pyexeci "filename"
pylexec "filename"
pylexeci "filename"
pyexecct ktrigger, "filename"
pylexect ktrigger, "filename"

pyinit

pyrun "statement"
pyruni "statement"
pylrn "statement"
pylrni "statement"
pyrunct ktrigger, "statement"
pylrnct ktrigger, "statement"

```

## Opcodes pour le Traitement d'Image.

```

iimagenum imagecreate iwidth, iheight

imagefree iimagenum

ared agreeen ablue imagegetpixel iimagenum, ax, ay
kred kgreen kblue imagegetpixel iimagenum, kx, ky

iimagenum imageload filename

imagesave iimagenum, filename

imagegetpixel iimagenum, ax, ay, ared, agreeen, ablue
imagegetpixel iimagenum, kx, ky, kred, kgreen, kblue

iwidth iheight imagesize iimagenum

```

## Divers.

```
modmatrix iresfn, isrcmodfn, isrcparmf, imodscale, inum_mod, \\  
inum_parm, kupdate  
  
ires system_i itrigr, Scmd, [inowait]  
kres system ktrigr, Scmd, [knowait]
```

## Utilitaires.

```
csound -U atsa [options] nomfichier_entree nomfichier_sortie  
  
cs [-OPTIONS] <nom> [OPTIONS DE CSOUND ... ]  
  
csb64enc [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]  
  
csound -U cvanal [options] nomfichier_entree nomfichier_sortie  
cvanal [options] nomfichier_entree nomfichier_sortie  
  
dnoise [options] -i ficref_bruit -o ficson_sortie ficson_entree  
  
envext [-options] fichierson  
csound -U envext [-options] fichierson  
  
extractor [OPTIONS ... ] fichierentree  
  
het_export fichier_het fichier_textecsv  
csound -U het_export fichier_het fichier_textecsv  
  
het_import fichier_textecsv fichier_het  
csound -U het_import fichier_textecsv fichier_het  
  
csound -U hetro [options] nomfichier_entree nomfichier_sortie  
hetro [options] nomfichier_entree nomfichier_sortie  
  
csound -U lpanal [options] nomfichier_entree nomfichier_sortie  
lpanal [options] nomfichier_entree nomfichier_sortie  
  
makecsd [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]  
  
mixer [OPTIONS ... ] fichier [[OPTIONS... ] fichier] ...  
  
pv_export fichier_pv fichier_texte_csv  
csound -U pv_export fichier_pv fichier_texte_csv  
  
pv_import fichier_texte_csv fichier_pv  
csound -U pv_import fichier_texte_csv fichier_pv  
  
csound -U pvanal [options] nomfic_entree nomfic_sortie  
pvanal [options] nomfic_entree nomfic_sortie  
  
csound -U pvlook [options] fichier_entree  
pvlook [options] fichier_entree  
  
scale [OPTIONS ... ] fichier  
  
sdif2ad [options] fichier_entree fichier_sortie  
  
csound -U sndinfo [options] fichierson ...  
sndinfo [options] fichierson ...  
  
srconv [options] fichier_entree
```

---

# Annexe A. Liste des exemples

## **Syntaxe de l'Orchestre : En-tête.**

*0dbfs.csd* [examples/0dbfs.csd]

*0dbfs-1.csd* [examples/0dbfs-1.csd]

## **Syntaxe de l'Orchestre : Bloc d'Instructions.**

*opcode\_example.csd* [examples/opcode\_example.csd]

*instr.csd* [examples/instr.csd]

*endin.csd* [examples/endin.csd]

## **Syntaxe de l'Orchestre : Macros.**

*include.csd* [examples/include.csd]

*define.csd* [examples/define.csd]

*define\_args.csd* [examples/define\_args.csd]

*define.csd* [examples/define.csd]

*define\_args.csd* [examples/define\_args.csd]

## **Générateurs de Signal : Synthèse/Resynthèse Additive.**

*adsynt.csd* [examples/adsynt.csd]

*hsboscil.csd* [examples/hsboscil.csd]

*hsboscil\_midi.csd* [examples/hsboscil\_midi.csd]

*adsyn.csd* [examples/adsyn.csd]

## **Générateurs de Signal : Oscillateurs Élémentaires.**

*oscilikt.csd* [examples/oscilikt.csd]

*poscil3.csd* [examples/poscil3.csd]

*poscil3.csd* [examples/poscil3.csd]

*osciliktp.csd* [examples/osciliktp.csd]

*vibrato.csd* [examples/vibrato.csd]

*poscil.csd* [examples/poscil.csd]

*oscil.csd* [examples/oscil.csd]

*oscils.csd* [examples/oscils.csd]



*oscili.csd* [examples/oscili.csd]

*lfo.csd* [examples/lfo.csd]

*oscil3.csd* [examples/oscil3.csd]

*vibr.csd* [examples/vibr.csd]

*oscbnk.csd* [examples/oscbnk.csd]

*oscilikts.csd* [examples/oscilikts.csd]

### **Générateurs de Signal : Oscillateurs à Spectre Dynamique.**

*buzz.csd* [examples/buzz.csd]

*mpulse.csd* [examples/mpulse.csd]

*vco2.csd* [examples/vco2.csd]

*vco.csd* [examples/vco.csd]

*gbuzz.csd* [examples/gbuzz.csd]

### **Générateurs de Signal : Synthèse FM.**

*fmwurlie.csd* [examples/fmwurlie.csd]

*fmvoice.csd* [examples/fmvoice.csd]

*fmbell.csd* [examples/fmbell.csd]

*fmmetal.csd* [examples/fmmetal.csd]

*fmrhode.csd* [examples/fmrhode.csd]

*fmpercfl.csd* [examples/fmpercfl.csd]

*crossfm.csd* [examples/crossfm.csd]

*fmb3.csd* [examples/fmb3.csd]

*foscil.csd* [examples/foscil.csd]

*foscili.csd* [examples/foscili.csd]

### **Générateurs de Signal : Synthèse Granulaire.**

*partikkel.csd* [examples/partikkel.csd]

*diskgrain.csd* [examples/diskgrain.csd]

*vosim.csd* [examples/vosim.csd]

*sndwarp.csd* [examples/sndwarp.csd]

*grain3.csd* [examples/grain3.csd]

*PartikkelExample1.csd* [examples/PartikkelExample1.csd]

*partikkel\_softsync.csd* [examples/partikkel\_softsync.csd]

*fof.csd* [examples/fof.csd]

*grain.csd* [examples/grain.csd]

*granule.csd* [examples/granule.csd]

*grain2.csd* [examples/grain2.csd]

*fof2.csd* [examples/fof2.csd]

*fof2-2.csd* [examples/fof2-2.csd]

### **Générateurs de Signal : Synthèse Hyper Vectorielle.**

*hvs1.csd* [examples/hvs1.csd]

*hvs2.csd* [examples/hvs2.csd]

*hvs2-2.csd* [examples/hvs2-2.csd]

*hvs3.csd* [examples/hvs3.csd]

### **Générateurs de Signal : Générateurs Linéaires et Exponentiels.**

*expsega.csd* [examples/expsega.csd]

*expsegr.csd* [examples/expsegr.csd]

*loopseg.csd* [examples/loopseg.csd]

*transegr.csd* [examples/transegr.csd]

*linseg.csd* [examples/linseg.csd]

*expon.csd* [examples/expon.csd]

*linsegr.csd* [examples/linsegr.csd]

*line.csd* [examples/line.csd]

*lpshold.csd* [examples/lpshold.csd]

*logcurve.csd* [examples/logcurve.csd]

*loopxseg.csd* [examples/loopxseg.csd]

*expseg.csd* [examples/expseg.csd]

*looptseg.csd* [examples/looptseg.csd]

*gainslider.csd* [examples/gainslider.csd]

*scale.csd* [examples/scale.csd]

*expcurve.csd* [examples/expcurve.csd]

*loopsegp.csd* [examples/loopsegp.csd]

*transeg.csd* [examples/transeg.csd]

### **Générateurs de Signal : Générateurs d'Enveloppe.**

*madsr.csd* [examples/madsr.csd]

*envlpx.csd* [examples/envlpx.csd]

*adsr.csd* [examples/adsr.csd]

### **Générateurs de Signal : Modèles et Emulations.**

*stix.csd* [examples/stix.csd]

*cabasa.csd* [examples/cabasa.csd]

*barmodel.csd* [examples/barmodel.csd]

*shaker.csd* [examples/shaker.csd]

*crunch.csd* [examples/crunch.csd]

*marimba.csd* [examples/marimba.csd]

*bamboo.csd* [examples/bamboo.csd]

*voice.csd* [examples/voice.csd]

*planet.csd* [examples/planet.csd]

*sekere.csd* [examples/sekere.csd]

*sandpaper.csd* [examples/sandpaper.csd]

*lorenz.csd* [examples/lorenz.csd]

*dripwater.csd* [examples/dripwater.csd]

*gogobel.csd* [examples/gogobel.csd]

*chuap.csd* [examples/chuap.csd]

*mandol.csd* [examples/mandol.csd]

*guiro.csd* [examples/guiro.csd]

*vibes.csd* [examples/vibes.csd]

*moog.csd* [examples/moog.csd]

*sleighbells.csd* [examples/sleighbells.csd]

*prepiano.csd* [examples/prepiano.csd]

*tambourine.csd* [examples/tambourine.csd]

**Générateurs de Signal : Phaseurs.**

*syncphasor.csd* [examples/syncphasor.csd]

*syncphasor-CZresonance.csd* [examples/syncphasor-CZresonance.csd]

*phasor.csd* [examples/phasor.csd]

*phasorbnk.csd* [examples/phasorbnk.csd]

**Générateurs de Signal : Générateurs de Nombres Aléatoires (de Bruit).**

*bexprnd.csd* [examples/bexprnd.csd]

*betarand.csd* [examples/betarand.csd]

*rnd31.csd* [examples/rnd31.csd]

*rnd31\_krate.csd* [examples/rnd31\_krate.csd]

*rnd31\_seed7.csd* [examples/rnd31\_seed7.csd]

*rnd31\_time.csd* [examples/rnd31\_time.csd]

*gauss.csd* [examples/gauss.csd]

*randomi.csd* [examples/randomi.csd]

*rand.csd* [examples/rand.csd]

*randi.csd* [examples/randi.csd]

*pcauchy.csd* [examples/pcauchy.csd]

*cauchy.csd* [examples/cauchy.csd]

*noise.csd* [examples/noise.csd]

*noise-2.csd* [examples/noise-2.csd]

*jitter.csd* [examples/jitter.csd]

*jitter2.csd* [examples/jitter2.csd]

*unirand.csd* [examples/unirand.csd]

*trirand.csd* [examples/trirand.csd]

*exprand.csd* [examples/exprand.csd]

*weibull.csd* [examples/weibull.csd]

*randomh.csd* [examples/randomh.csd]

*trandom.csd* [examples/trandom.csd]

*pinkish.csd* [examples/pinkish.csd]

*linrand.csd* [examples/linrand.csd]

*randh.csd* [examples/randh.csd]

*random.csd* [examples/random.csd]

*poisson.csd* [examples/poisson.csd]

### **Générateurs de Signal : Reproduction de Sons Echantillonnés.**

*loscil.csd* [examples/loscil.csd]

*lphasor.csd* [examples/lphasor.csd]

*lposcilsa2.csd* [examples/lposcilsa2.csd]

*bbcutm.csd* [examples/bbcutm.csd]

*lposcila.csd* [examples/lposcila.csd]

*fluidcomplex.csd* [examples/fluidcomplex.csd]

*waveset.csd* [examples/waveset.csd]

*lposcilsa.csd* [examples/lposcilsa.csd]

*fluidcomplex.csd* [examples/fluidcomplex.csd]

*loscil3.csd* [examples/loscil3.csd]

### **Générateurs de Signal : Synthèse par Balayage.**

*scans.csd* [examples/scans.csd]

*scantable.csd* [examples/scantable.csd]

### **Générateurs de Signal : Accès aux Tables.**

*table.csd* [examples/table.csd]

### **Générateurs de Signal : Synthèse par Terrain d'Ondes.**

*wterrain.csd* [examples/wterrain.csd]

### **Générateurs de Signal : Modèles Physiques par Guide d'Onde.**

*wgbow.csd* [examples/wgbow.csd]

*wgclar.csd* [examples/wgclar.csd]

*streson.csd* [examples/streson.csd]

*repluck.csd* [examples/repluck.csd]

*wgpluck2.csd* [examples/wgpluck2.csd]

*wgpluck.csd* [examples/wgpluck.csd]

*wgpluck\_brighter.csd* [examples/wgpluck\_brighter.csd]

*wgbowedbar.csd* [examples/wgbowedbar.csd]

*wgbrass.csd* [examples/wgbrass.csd]

*wgflute.csd* [examples/wgflute.csd]

*pluck.csd* [examples/pluck.csd]

**E/S de Signal : E/S Fichier.**

*fout.csd* [examples/fout.csd]

*fout\_poly.csd* [examples/fout\_poly.csd]

*fout\_ftable.csd* [examples/fout\_ftable.csd]

*fprints.csd* [examples/fprints.csd]

*fprintks.csd* [examples/fprintks.csd]

*fprintks-2.csd* [examples/fprintks-2.csd]

*scogen.csd* [examples/scogen.csd]

*readk.csd* [examples/readk.csd]

*dumpk.csd* [examples/dumpk.csd]

**E/S de Signal : Entrée de Signal.**

*diskin.csd* [examples/diskin.csd]

*soundin.csd* [examples/soundin.csd]

*diskin2.csd* [examples/diskin2.csd]

*inrg.csd* [examples/inrg.csd]

**E/S de Signal : Sortie de Signal.**

*mdelay.csd* [examples/mdelay.csd]

*outrg.csd* [examples/outrg.csd]

**E/S de Signal : Impression et Affichage.**

*printk.csd* [examples/printk.csd]

*display.csd* [examples/display.csd]

*printks.csd* [examples/printks.csd]

*printk2.csd* [examples/printk2.csd]

*prints.csd* [examples/prints.csd]

*print.csd* [examples/print.csd]

*flashtxt.csd* [examples/flashtxt.csd]

*dispfft.csd* [examples/disppfft.csd]

### **E/S de Signal : Requêtes sur les Fichiers Sons.**

*filesr.csd* [examples/filesr.csd]

*filelen.csd* [examples/filelen.csd]

*filepeak.csd* [examples/filepeak.csd]

*filebit.csd* [examples/filebit.csd]

*filenchnls.csd* [examples/filenchnls.csd]

### **Modificateurs de Signal : Modificateurs d'Amplitude.**

*dam.csd* [examples/dam.csd]

*dam\_expanded.csd* [examples/dam\_expanded.csd]

*gain.csd* [examples/gain.csd]

*clip.csd* [examples/clip.csd]

*balance.csd* [examples/balance.csd]

### **Modificateurs de Signal : Convolution et Morphing.**

*ftconv.csd* [examples/ftconv.csd]

*cross2.csd* [examples/cross2.csd]

*ftmorf.csd* [examples/ftmorf.csd]

*dconv.csd* [examples/dconv.csd]

### **Modificateurs de Signal : Retard.**

*delayw.csd* [examples/delayw.csd]

*delay.csd* [examples/delay.csd]

### **Modificateurs de Signal : Panning et Spatialisation.**

*bformenc.csd* [examples/bformenc.csd]

*vbap8move.csd* [examples/vbap8move.csd]

*hrtfer.csd* [examples/hrtfer.csd]

*hrtfstat.csd* [examples/hrtfstat.csd]

*bformenc1.csd* [examples/bformenc1.csd]  
*spat3d\_stereo.csd* [examples/spat3d\_stereo.csd]  
*spat3d\_UHJ.csd* [examples/spat3d\_UHJ.csd]  
*spat3d\_quad.csd* [examples/spat3d\_quad.csd]  
*hrtfmove2.csd* [examples/hrtfmove2.csd]  
*bformenc.csd* [examples/bformenc.csd]  
*bformenc1.csd* [examples/bformenc1.csd]  
*vbap8.csd* [examples/vbap8.csd]  
*hrtfmove.csd* [examples/hrtfmove.csd]

**Modificateurs de Signal : Réverbération.**

*freeverb.csd* [examples/freeverb.csd]  
*babo.csd* [examples/babo.csd]  
*babo\_expert.csd* [examples/babo\_expert.csd]  
*alpass.csd* [examples/alpass.csd]  
*comb.csd* [examples/comb.csd]  
*vcomb.csd* [examples/vcomb.csd]  
*nestedap.csd* [examples/nestedap.csd]  
*reverb.csd* [examples/reverb.csd]  
*reverb\_sc.csd* [examples/reverb\_sc.csd]  
*nreverb.csd* [examples/nreverb.csd]  
*nreverb\_ftable.csd* [examples/nreverb\_ftable.csd]

**Modificateurs de Signal : Opérateurs du Niveau Echantillon.**

*integ.csd* [examples/integ.csd]  
*denorm.csd* [examples/denorm.csd]  
*diff.csd* [examples/diff.csd]  
*opa.csd* [examples/opa.csd]  
*downsamp.csd* [examples/downsamp.csd]  
*interp.csd* [examples/interp.csd]  
*fold.csd* [examples/fold.csd]  
*vaset.csd* [examples/vaset.csd]



*vaget.csd* [examples/vaget.csd]

**Modificateurs de Signal : Effets Spéciaux.**

*flanger.csd* [examples/flanger.csd]

*distort1.csd* [examples/distort1.csd]

*harmon.csd* [examples/harmon.csd]

*phaser2.csd* [examples/phaser2.csd]

*phaser1.csd* [examples/phaser1.csd]

**Modificateurs de Signal : Filtres Standard.**

*mode.csd* [examples/mode.csd]

*atone.csd* [examples/atone.csd]

*butterlp.csd* [examples/butterlp.csd]

*biquad.csd* [examples/biquad.csd]

*biquad-2.csd* [examples/biquad-2.csd]

*clfilt\_lowpass.csd* [examples/clfilt\_lowpass.csd]

*clfilt\_highpass.csd* [examples/clfilt\_highpass.csd]

*butterbr.csd* [examples/butterbr.csd]

*butterbp.csd* [examples/butterbp.csd]

*doppler.csd* [examples/doppler.csd]

*butterhp.csd* [examples/butterhp.csd]

**Modificateurs de Signal : Filtres Standard : Résonants.**

*lpf18.csd* [examples/lpf18.csd]

*lowpass2.csd* [examples/lowpass2.csd]

*bqrez.csd* [examples/bqrez.csd]

*lowresx.csd* [examples/lowresx.csd]

*reson.csd* [examples/reson.csd]

*svfilter.csd* [examples/svfilter.csd]

*moogvcf2.csd* [examples/moogvcf2.csd]

*tbvcf.csd* [examples/tbvcf.csd]

*vlowres.csd* [examples/vlowres.csd]

*areson.csd* [examples/areson.csd]

*resonr.csd* [examples/resonr.csd]

*rezzy.csd* [examples/rezzy.csd]

*lowres.csd* [examples/lowres.csd]

*resony.csd* [examples/resony.csd]

*moogvcf.csd* [examples/moogvcf.csd]

### **Modificateurs de Signal : Filtres Standard : Contrôle.**

*portk.csd* [examples/portk.csd]

### **Modificateurs de Signal : Filtres Spécialisés.**

*pareq.csd* [examples/pareq.csd]

*rbjeq.csd* [examples/rbjeq.csd]

*hilbert.csd* [examples/hilbert.csd]

*hilbert\_barberpole.csd* [examples/hilbert\_barberpole.csd]

*dcblock.csd* [examples/dcblock.csd]

### **Modificateurs de Signal : Guides d'Onde.**

*wguide1.csd* [examples/wguide1.csd]

*wguide2.csd* [examples/wguide2.csd]

### **Modificateurs de Signal : Distorsion Non-Linéaire.**

*chebyshevpoly.csd* [examples/chebyshevpoly.csd]

*pdhalfy.csd* [examples/pdhalfy.csd]

*pdclip.csd* [examples/pdclip.csd]

*pdhalf.csd* [examples/pdhalf.csd]

*powershape.csd* [examples/powershape.csd]

### **Contrôle d'Instrument : Valeurs Conditionnelles.**

*lessequal.csd* [examples/lessequal.csd]

*greaterthan.csd* [examples/greaterthan.csd]

*equals.csd* [examples>equals.csd]

*greaterequal.csd* [examples/greaterequal.csd]

*lessthan.csd* [examples/lessthan.csd]

*notequal.csd* [examples/notequal.csd]

**Contrôle d'Instrument : Contrôle de Durée.**

*ihold.csd* [examples/ihold.csd]

*turnoff.csd* [examples/turnoff.csd]

**Contrôle d'Instrument : Appel d'Instrument.**

*schedkwhen.csd* [examples/schedkwhen.csd]

*schedwhen.csd* [examples/schedwhen.csd]

*mute.csd* [examples/mute.csd]

*schedule.csd* [examples/schedule.csd]

*event.csd* [examples/event.csd]

*event\_named.csd* [examples/event\_named.csd]

*schedkwhennamed.csd* [examples/schedkwhennamed.csd]

**Contrôle d'Instrument : Contrôle Séquentiel d'un Programme.**

*cgoto.csd* [examples/cgoto.csd]

*ckgoto.csd* [examples/ckgoto.csd]

*goto.csd* [examples/goto.csd]

*cigoto.csd* [examples/cigoto.csd]

*igoto.csd* [examples/igoto.csd]

*kgoto.csd* [examples/kgoto.csd]

*ifthen.csd* [examples/ifthen.csd]

*cngoto.csd* [examples/cngoto.csd]

*igoto.csd* [examples/igoto.csd]

*kgoto.csd* [examples/kgoto.csd]

**Contrôle d'Instrument : Contrôle de l'Exécution en Temps Réel.**

*maxalloc.csd* [examples/maxalloc.csd]

*prealloc.csd* [examples/prealloc.csd]

*jacktransport.csd* [examples/jacktransport.csd]

*active.csd* [examples/active.csd]

*active\_k.csd* [examples/active\_k.csd]

*active\_scale.csd* [examples/active\_scale.csd]

*cpuprc.csd* [examples/cpuprc.csd]

### **Contrôle d'Instrument : Initialisation et Réinitialisation.**

*reinit.csd* [examples/reinit.csd]

*assign.csd* [examples/assign.csd]

*p.csd* [examples/p.csd]

*reinit.csd* [examples/reinit.csd]

### **Contrôle d'Instrument : Détection et Contrôle.**

*metro.csd* [examples/metro.csd]

*tempo.csd* [examples/tempo.csd]

*xyin.csd* [examples/xyin.csd]

*wii.csd* [examples/wii.csd]

*pitchamdf.csd* [examples/pitchamdf.csd]

*tempoval.csd* [examples/tempoval.csd]

*follow2.csd* [examples/follow2.csd]

*pindex.csd* [examples/pindex.csd]

*pitch.csd* [examples/pitch.csd]

*ptrack.csd* [examples/ptrack.csd]

*sensekey.csd* [examples/sensekey.csd]

*FLpanel-sensekey.csd* [examples/FLpanel-sensekey.csd]

*FLpanel-sensekey2.csd* [examples/FLpanel-sensekey2.csd]

*seqtime.csd* [examples/seqtime.csd]

*follow.csd* [examples/follow.csd]

*p5g.csd* [examples/p5g.csd]

*pcount.csd* [examples/pcount.csd]

*peak.csd* [examples/peak.csd]

*changed.csd* [examples/changed.csd]

*setctrl.csd* [examples/setctrl.csd]

*trigger.csd* [examples/trigger.csd]

*checkbox.csd* [examples/checkbox.csd]

*rms.csd* [examples/rms.csd]

*tempest.csd* [examples/tempest.csd]

**Contrôle d'Instrument : Contrôle de sous-instrument.**

*subinstr.csd* [examples/subinstr.csd]

*subinstr\_named.csd* [examples/subinstr\_named.csd]

**Contrôle d'Instrument : Lecture du Temps.**

*date.csd* [examples/date.csd]

*timeinsts.csd* [examples/timeinsts.csd]

*timek.csd* [examples/timek.csd]

*times.csd* [examples/times.csd]

*rtclock.csd* [examples/rtclock.csd]

*readclock.csd* [examples/readclock.csd]

*timeinstk.csd* [examples/timeinstk.csd]

*date.csd* [examples/date.csd]

**Contrôle des Tables de Fonction.**

*ftgen.csd* [examples/ftgen.csd]

*ftgen-2.csd* [examples/ftgen-2.csd]

*ftgentmp.csd* [examples/ftgentmp.csd]

**Contrôle des Tables de Fonction : Requêtes sur une Table.**

*ftsr.csd* [examples/ftsr.csd]

*ftlptim.csd* [examples/ftlptim.csd]

*tableng.csd* [examples/tableng.csd]

*ftchnls.csd* [examples/ftchnls.csd]

*nsamp.csd* [examples/nsamp.csd]

*ftlen.csd* [examples/ftlen.csd]

**Contrôle des Tables de Fonction : Sélection Dynamique.**

*tablexkt.csd* [examples/tablexkt.csd]

**Contrôle des Tables de Fonction : Opérations de Lecture/Ecriture.**

*tabmorphi.csd* [examples/tabmorphi.csd]

*tabmorph.csd* [examples/tabmorph.csd]

*ftsavc.csd* [examples/ftsavc.csd]

*tabmorphak.csd* [examples/tabmorphak.csd]

*tabmorpha.csd* [examples/tabmorpha.csd]

**FLTK : Conteneurs.**

*FLscroll.csd* [examples/FLscroll.csd]

*FLtabs.csd* [examples/FLtabs.csd]

*FLpanel.csd* [examples/FLpanel.csd]

**FLTK : Valueurs.**

*FLroller.csd* [examples/FLroller.csd]

*FLtext.csd* [examples/FLtext.csd]

*FLslider.csd* [examples/FLslider.csd]

*FLslider-2.csd* [examples/FLslider-2.csd]

*FLknob.csd* [examples/FLknob.csd]

*FLknob-2.csd* [examples/FLknob-2.csd]

*FLjoy.csd* [examples/FLjoy.csd]

*FLcount.csd* [examples/FLcount.csd]

**FLTK : Autres.**

*FLvslidBnk2.csd* [examples/FLvslidBnk2.csd]

*FLslidBnkSet.csd* [examples/FLslidBnkSet.csd]

*FLhvsBox.csd* [examples/FLhvsBox.csd]

*FLbutBank.csd* [examples/FLbutBank.csd]

*FLmouse.csd* [examples/FLmouse.csd]

*FLsavesnap\_simple.csd* [examples/FLsavesnap\_simple.csd]

*FLsavesnap.csd* [examples/FLsavesnap.csd]

*FLhvsBoxSetValue.csd* [examples/FLhvsBoxSetValue.csd]

*FLxyin.csd* [examples/FLxyin.csd]

*FLxyin-2.csd* [examples/FLxyin-2.csd]

*FLslidBnkGetHandle.csd* [examples/FLslidBnkGetHandle.csd]

*FLvslidBnk.csd* [examples/FLvslidBnk.csd]

*FLslidBnk2Setk.csd* [examples/FLslidBnk2Setk.csd]

*FLslidBnk.csd* [examples/FLslidBnk.csd]

*FLbox.csd* [examples/FLbox.csd]

*FLexecButton.csd* [examples/FLexecButton.csd]

*FLkeyIn.csd* [examples/FLkeyIn.csd]

*FLslidBnk2Set.csd* [examples/FLslidBnk2Set.csd]

*FLbutton.csd* [examples/FLbutton.csd]

*FLslidBnk2.csd* [examples/FLslidBnk2.csd]

*FLslidBnkSetk.csd* [examples/FLslidBnkSetk.csd]

*vphaseseg.csd* [examples/vphaseseg.csd]

*FLvalue.csd* [examples/FLvalue.csd]

### **FLTK : Apparence.**

*FLsetcolor.csd* [examples/FLsetcolor.csd]

*FLsetText.csd* [examples/FLsetText.csd]

### **Opérations Mathématiques : Compérateurs et Accumulateurs.**

*multiplies.csd* [examples/multiplies.csd]

*bitwise.csd* [examples/bitwise.csd]

*bitshift.csd* [examples/bitshift.csd]

*subtracts.csd* [examples/subtracts.csd]

*adds.csd* [examples/adds.csd]

*raises.csd* [examples/raises.csd]

*modulus.csd* [examples/modulus.csd]

*divides.csd* [examples/divides.csd]

### **Opérations Mathématiques : Fonctions Mathématiques.**

*log.csd* [examples/log.csd]

*powoftwo.csd* [examples/powoftwo.csd]

*logbtwo.csd* [examples/logbtwo.csd]

*int.csd* [examples/int.csd]

*sqr.csd* [examples/sqr.csd]

*log10.csd* [examples/log10.csd]

*frac.csd* [examples/frac.csd]

*exp.csd* [examples/exp.csd]

*abs.csd* [examples/abs.csd]

### **Opérations Mathématiques : Fonctions Trigonométriques.**

*tan.csd* [examples/tan.csd]

*taninv.csd* [examples/taninv.csd]

*cosh.csd* [examples/cosh.csd]

*sinh.csd* [examples/sinh.csd]

*cos.csd* [examples/cos.csd]

*tanh.csd* [examples/tanh.csd]

*cosinv.csd* [examples/cosinv.csd]

*sininv.csd* [examples/sininv.csd]

*sin.csd* [examples/sin.csd]

### **Opérations Mathématiques : Fonctions d'Amplitude.**

*dbamp.csd* [examples/dbamp.csd]

*db.csd* [examples/db.csd]

*dbfsamp.csd* [examples/dbfsamp.csd]

*ampdbfs.csd* [examples/ampdbfs.csd]

*ampdb.csd* [examples/ampdb.csd]

### **Opérations Mathématiques : Fonctions aléatoires.**

*rnd.csd* [examples/rnd.csd]

*birnd.csd* [examples/birnd.csd]

### **Opérations Mathématiques : Opcodes Equivalents à des Fonctions.**

*divz.csd* [examples/divz.csd]

*pow.csd* [examples/pow.csd]



*polynomial.csd* [examples/polynomial.csd]

*taninv2.csd* [examples/taninv2.csd]

### **Conversion des Hauteurs : Fonctions.**

*cpspch.csd* [examples/cpspch.csd]

*semitone.csd* [examples/semitone.csd]

*octpch.csd* [examples/octpch.csd]

*cpsmidinn.csd* [examples/cpsmidinn.csd]

*octave.csd* [examples/octave.csd]

*cpsoct.csd* [examples/cpsoct.csd]

*cpsmidinn.csd* [examples/cpsmidinn.csd]

*octcps.csd* [examples/octcps.csd]

*cent.csd* [examples/cent.csd]

*pchoct.csd* [examples/pchoct.csd]

*cpsmidinn.csd* [examples/cpsmidinn.csd]

### **Conversion des Hauteurs : Opcodes de Hauteurs.**

*cps2pch.csd* [examples/cps2pch.csd]

*cps2pch\_ftable.csd* [examples/cps2pch\_ftable.csd]

*cps2pch\_19et.csd* [examples/cps2pch\_19et.csd]

*cpsxpch.csd* [examples/cpsxpch.csd]

*cpsxpch\_105et.csd* [examples/cpsxpch\_105et.csd]

*cpsxpch\_pierce.csd* [examples/cpsxpch\_pierce.csd]

*cpstun.csd* [examples/cpstun.csd]

*cpstuni.csd* [examples/cpstuni.csd]

### **MIDI en Temps-Réel : Entrée.**

*pgmassign.csd* [examples/pgmassign.csd]

*pgmassign\_ignore.csd* [examples/pgmassign\_ignore.csd]

*pgmassign\_advanced.csd* [examples/pgmassign\_advanced.csd]

*aftouch.csd* [examples/aftouch.csd]

*polyaft.csd* [examples/polyaft.csd]

*veloc.csd* [examples/veloc.csd]

*pchbend.csd* [examples/pchbend.csd]

*notnum.csd* [examples/notnum.csd]

*notnum\_complex.csd* [examples/notnum\_complex.csd]

### **MIDI en Temps-Réel : Sortie.**

*outkpc.csd* [examples/outkpc.csd]

*outkpc\_fltk.csd* [examples/outkpc\_fltk.csd]

### **MIDI en Temps-Réel : Convertisseurs.**

*octmidi.csd* [examples/octmidi.csd]

*octmidib.csd* [examples/octmidib.csd]

*pchmidib.csd* [examples/pchmidib.csd]

*cpsmidib.csd* [examples/cpsmidib.csd]

*cpsmidi.csd* [examples/cpsmidi.csd]

*cpstmid.csd* [examples/cpstmid.csd]

*pchmidi.csd* [examples/pchmidi.csd]

*ampmidi.csd* [examples/ampmidi.csd]

### **MIDI en Temps-Réel : E/S Génériques.**

*midiin.csd* [examples/midiin.csd]

### **MIDI en Temps-Réel : Extension d'Evènements.**

*xtratim.csd* [examples/xtratim.csd]

*xtratim-2.csd* [examples/xtratim-2.csd]

### **MIDI en Temps-Réel : Sortie de Note.**

*noteondur.csd* [examples/noteondur.csd]

*moscil.csd* [examples/moscil.csd]

*midion\_simple.csd* [examples/midion\_simple.csd]

*midion\_scale.csd* [examples/midion\_scale.csd]

*noteondur2.csd* [examples/noteondur2.csd]

### **MIDI en Temps-Réel : Interopérabilité MIDI/Partition.**

*midinoteoff.csd* [examples/midinoteoff.csd]  
*midichannelaftertouch.csd* [examples/midichannelaftertouch.csd]  
*midinoteonoct.csd* [examples/midinoteonoct.csd]  
*midinoteonkey.csd* [examples/midinoteonkey.csd]  
*midichn.csd* [examples/midichn.csd]  
*midichn\_advanced.csd* [examples/midichn\_advanced.csd]  
*midinoteonpch.csd* [examples/midinoteonpch.csd]  
*midipitchbend.csd* [examples/midipitchbend.csd]  
*midinoteoncps.csd* [examples/midinoteoncps.csd]

**Traitement Spectral : Streaming.**

*pvsmaska.csd* [examples/pvsmaska.csd]  
*pvsbin.csd* [examples/pvsbin.csd]  
*pvscompress.csd* [examples/pvscompress.csd]  
*pvsfreeze.csd* [examples/pvsfreeze.csd]  
*pvscent.csd* [examples/pvscent.csd]  
*pvsosc.csd* [examples/pvsosc.csd]  
*pvsfwrite.csd* [examples/pvsfwrite.csd]  
*pvsmorph.csd* [examples/pvsmorph.csd]  
*pvsmorph2.csd* [examples/pvsmorph2.csd]  
*pvsblur.csd* [examples/pvsblur.csd]  
*pvsdisp.csd* [examples/pvsdisp.csd]  
*pvsarp.csd* [examples/pvsarp.csd]  
*pvsarp2.csd* [examples/pvsarp2.csd]  
*pvsbandr.csd* [examples/pvsbandr.csd]  
*pvsadsyn.csd* [examples/pvsadsyn.csd]  
*pvsynth.csd* [examples/pvsynth.csd]  
*pvsbandp.csd* [examples/pvsbandp.csd]  
*pvsfilter.csd* [examples/pvsfilter.csd]  
*pvsbufread.csd* [examples/pvsbufread.csd]  
*pvscale.csd* [examples/pvscale.csd]

*pvspitch.csd* [examples/pvspitch.csd]

*pvshift.csd* [examples/pvshift.csd]

### **Chaînes : Définition.**

*strset.csd* [examples/strset.csd]

### **Chaînes : Manipulation.**

*sprintfk.csd* [examples/sprintfk.csd]

*strsub.csd* [examples/strsub.csd]

### **Chaînes : Conversion.**

*strtod.csd* [examples/strtod.csd]

*strtol.csd* [examples/strtol.csd]

*strtodk.csd* [examples/strtodk.csd]

*strtolk.csd* [examples/strtolk.csd]

### **Vectorel : Tableaux.**

*vtablek.csd* [examples/vtablek.csd]

*vtable1k.csd* [examples/vtable1k.csd]

*vtablei.csd* [examples/vtablei.csd]

*vtablewa.csd* [examples/vtablewa.csd]

*vtablewk.csd* [examples/vtablewk.csd]

### **Vectorel : Opérations Scalaires.**

*vmult-2.csd* [examples/vmult-2.csd]

*vmult.csd* [examples/vmult.csd]

*vexp\_i.csd* [examples/vexp\_i.csd]

*vpow\_i.csd* [examples/vpow\_i.csd]

*vadd.csd* [examples/vadd.csd]

*vmult\_i.csd* [examples/vmult\_i.csd]

*vadd\_i.csd* [examples/vadd\_i.csd]

*vpow.csd* [examples/vpow.csd]

*vexp.csd* [examples/vexp.csd]

**Vectoriel : Opérations Vectorielles.**

*vcopy.csd* [examples/vcopy.csd]

*vaddv.csd* [examples/vaddv.csd]

*vmultv.csd* [examples/vmultv.csd]

*vexpv.csd* [examples/vexpv.csd]

*vmap.csd* [examples/vmap.csd]

*vsubv.csd* [examples/vsubv.csd]

*vdivv.csd* [examples/vdivv.csd]

*vpowv.csd* [examples/vpowv.csd]

**Vectoriel : Enveloppes.**

*vexpseg.csd* [examples/vexpseg.csd]

*vlinseg.csd* [examples/vlinseg.csd]

**Vectoriel : Aléatoire.**

*vrandh.csd* [examples/vrandh.csd]

*vrandi.csd* [examples/vrandi.csd]

**Vectoriel : Automates Cellulaires.**

*vcella.csd* [examples/vcella.csd]

**Système de Patch Zak.**

*zkw.csd* [examples/zkw.csd]

*zkmod.csd* [examples/zkmod.csd]

*zaw.csd* [examples/zaw.csd]

*ziw.csd* [examples/ziw.csd]

*zar.csd* [examples/zar.csd]

*zir.csd* [examples/zir.csd]

*zamod.csd* [examples/zamod.csd]

*zarg.csd* [examples/zarg.csd]

*zkcl.csd* [examples/zkcl.csd]

*ziwm.csd* [examples/ziwm.csd]

*zkwm.csd* [examples/zkwm.csd]

*zakinit.csd* [examples/zakinit.csd]

*zacl.csd* [examples/zacl.csd]

*zawm.csd* [examples/zawm.csd]

*zkr.csd* [examples/zkr.csd]

### **Accueil de Plugin : DSSI et LADSPA.**

*dssi4cs.csd* [examples/dssi4cs.csd]

### **Accueil de Plugin : VST.**

*vst4cs.csd* [examples/vst4cs.csd]

*vst4cs.csd* [examples/vst4cs.csd]

*vst4cs.csd* [examples/vst4cs.csd]

*vst4cs.csd* [examples/vst4cs.csd]

*vst4cs.csd* [examples/vst4cs.csd]

*vst4cs.csd* [examples/vst4cs.csd]

### **OSC.**

*OSCmidisend.csd* [examples/OSCmidisend.csd]

*OSCmidircv.csd* [examples/OSCmidircv.csd]

### **Opcodes pour le Traitement à Distance.**

*midremot.csd* [examples/midremot.csd]

*insremot.csd* [examples/insremot.csd]

*insremotM.csd* [examples/insremotM.csd]

### **Opcodes pour le Traitement d'Image.**

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodesdemo2.csd* [examples/imageopcodesdemo2.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

**Divers.**

*system.csd* [examples/system.csd]

*modmatrix.csd* [examples/modmatrix.csd]

# Annexe B. Conversion de Hauteur

**Tableau B.1. Conversion de Hauteur**

Note (anglais)	Note (français)	Hz	cpspch
C-1	do-2	8.176	3.00
C#-1	do#-2	8.662	3.01
D-1	ré-2	9.177	3.02
D#-1	ré#-2	9.723	3.03
E-1	mi-2	10.301	3.04
F-1	fa-2	10.913	3.05
F#-1	fa#-2	11.562	3.06
G-1	sol-2	12.250	3.07
G#-1	sol#-2	12.978	3.08
A-1	la-2	13.750	3.09
A#-1	la#-2	14.568	3.10
B-1	si-2	15.434	3.11
C0	do-1	16.352	4.00
C#0	do#-1	17.324	4.01
D0	ré-1	18.354	4.02
D#0	ré#-1	19.445	4.03
E0	mi-1	20.602	4.04
F0	fa-1	21.827	4.05
F#0	fa#-1	23.125	4.06
G0	sol-1	24.500	4.07
G#0	sol#-1	25.957	4.08
A0	la-1	27.500	4.09
A#0	la#-1	29.135	4.10
B0	si-1	30.868	4.11
C1	do0	32.703	5.00
C#1	do#0	34.648	5.01
D1	ré0	36.708	5.02
D#1	ré#0	38.891	5.03
E1	mi0	41.203	5.04
F1	fa0	43.654	5.05
F#1	fa#0	46.249	5.06
G1	sol0	48.999	5.07
G#1	sol#0	51.913	5.08
A1	la0	55.000	5.09
A#1	la#0	58.270	5.10
B1	si0	61.735	5.11



Note (anglais)	Note (français)	Hz	cpSPch
C2	do1	65.406	6.00
C#2	do#1	69.296	6.01
D2	ré1	73.416	6.02
D#2	ré#1	77.782	6.03
E2	mi1	82.407	6.04
F2	fa1	87.307	6.05
F#2	fa#1	92.499	6.06
G2	sol1	97.999	6.07
G#2	sol#1	103.826	6.08
A2	la1	110.000	6.09
A#2	la#1	116.541	6.10
B2	si1	123.471	6.11
C3	do2	130.813	7.00
C#3	do#2	138.591	7.01
D3	ré2	146.832	7.02
D#3	ré#2	155.563	7.03
E3	mi2	164.814	7.04
F3	fa2	174.614	7.05
F#3	fa#2	184.997	7.06
G3	sol2	195.998	7.07
G#3	sol#2	207.652	7.08
A3	la2	220.000	7.09
A#3	la#2	233.082	7.10
B3	si2	246.942	7.11
C4	do3	261.626	8.00
C#4	do#3	277.183	8.01
D4	ré3	293.665	8.02
D#4	ré#3	311.127	8.03
E4	mi3	329.628	8.04
F4	fa3	349.228	8.05
F#4	fa#3	369.994	8.06
G4	sol3	391.995	8.07
G#4	sol#3	415.305	8.08
A4	la3	440.000	8.09
A#4	la#3	466.164	8.10
B4	si3	493.883	8.11
C5	do4	523.251	9.00
C#5	do#4	554.365	9.01
D5	ré4	587.330	9.02
D#5	ré#4	622.254	9.03
E5	mi4	659.255	9.04

Note (anglais)	Note (français)	Hz	cpspch
F5	fa4	698.456	9.05
F#5	fa#4	739.989	9.06
G5	sol4	783.991	9.07
G#5	sol#4	830.609	9.08
A5	la4	880.000	9.09
A#5	la#4	932.328	9.10
B5	si4	987.767	9.11
C6	do5	1046.502	10.00
C#6	do#5	1108.731	10.01
D6	ré5	1174.659	10.02
D#6	ré#5	1244.508	10.03
E6	mi5	1318.510	10.04
F6	fa5	1396.913	10.05
F#6	fa#5	1479.978	10.06
G6	sol5	1567.982	10.07
G#6	sol#5	1661.219	10.08
A6	la5	1760.000	10.09
A#6	la#5	1864.655	10.10
B6	si5	1975.533	10.11
C7	do6	2093.005	11.00
C#7	do#6	2217.461	11.01
D7	ré6	2349.318	11.02
D#7	ré#6	2489.016	11.03
E7	mi6	2637.020	11.04
F7	fa6	2793.826	11.05
F#7	fa#6	2959.955	11.06
G7	sol6	3135.963	11.07
G#7	sol#6	3322.438	11.08
A7	la6	3520.000	11.09
A#7	la#6	3729.310	11.10
B7	si6	3951.066	11.11
C8	do7	4186.009	12.00
C#8	do#7	4434.922	12.01
D8	ré7	4698.636	12.02
D#8	ré#7	4978.032	12.03
E8	mi7	5274.041	12.04
F8	fa7	5587.652	12.05
F#8	fa#7	5919.911	12.06
G8	sol7	6271.927	12.07
G#8	sol#7	6644.875	12.08
A8	la7	7040.000	12.09

Note (anglais)	Note (français)	Hz	cpspch
A#8	la#7	7458.620	12.10
B8	si7	7902.133	12.11
C9	do8	8372.018	13.00
C#9	do#8	8869.844	13.01
D9	ré8	9397.273	13.02
D#9	ré#8	9956.063	13.03
E9	mi8	10548.08	13.04
F9	fa8	11175.30	13.05
F#9	fa#8	11839.82	13.06
G9	sol8	12543.85	13.07

---

# Annexe C. Valeurs d'Intensité du Son

**Tableau C.1. Valeurs d'Intensité du Son (pour un ton pur à 1000 Hz)**

Dynamiques	Intensité (W/m <sup>2</sup> )	Niveau (dB)
douleur	1	120
fff	10 <sup>-2</sup>	100
f	10 <sup>-4</sup>	80
p	10 <sup>-6</sup>	60
ppp	10 <sup>-8</sup>	40
seuil d'audibilité	10 <sup>-12</sup>	0

---

# Annexe D. Valeurs de Formant

**Tableau D.1. alto « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	800	1150	2800	3500	4950
amp (dB)	0	-4	-20	-36	-60
larg. bande (Hz)	80	90	120	130	140

**Tableau D.2. alto « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1600	2700	3300	4950
amp (dB)	0	-24	-30	-35	-60
larg. bande (Hz)	60	80	120	150	200

**Tableau D.3. alto « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	1700	2700	3700	4950
amp (dB)	0	-20	-30	-36	-60
larg. bande (Hz)	50	100	120	150	200

**Tableau D.4. alto « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	450	800	2830	3500	4950
amp (dB)	0	-9	-16	-28	-55
larg. bande (Hz)	70	80	100	130	135

**Tableau D.5. alto « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	325	700	2530	3500	4950
amp (dB)	0	-12	-30	-40	-64
larg. bande (Hz)	50	60	170	180	200

**Tableau D.6. basse « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	600	1040	2250	2450	2750
amp (dB)	0	-7	-9	-9	-20
larg. bande (Hz)	60	70	110	120	130

**Tableau D.7. basse « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1620	2400	2800	3100
amp (dB)	0	-12	-9	-12	-18
larg. bande (Hz)	40	80	100	120	120

**Tableau D.8. basse « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	250	1750	2600	3050	3340
amp (dB)	0	-30	-16	-22	-28
larg. bande (Hz)	60	90	100	120	120

**Tableau D.9. basse « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	750	2400	2600	2900
amp (dB)	0	-11	-21	-20	-40
larg. bande (Hz)	40	80	100	120	120

**Tableau D.10. basse « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	600	2400	2675	2950
amp (dB)	0	-20	-32	-28	-36
larg. bande (Hz)	40	80	100	120	120

**Tableau D.11. haute-contre « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	660	1120	2750	3000	3350
amp (dB)	0	-6	-23	-24	-38
larg. bande (Hz)	80	90	120	130	140

**Tableau D.12. haute-contre « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	440	1800	2700	3000	3300
amp (dB)	0	-14	-18	-20	-20
larg. bande (Hz)	70	80	100	120	120

**Tableau D.13. haute-contre « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	270	1850	2900	3350	3590
amp (dB)	0	-24	-24	-36	-36
larg. bande (Hz)	40	90	100	120	120

**Tableau D.14. haute-contre « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	430	820	2700	3000	3300
amp (dB)	0	-10	-26	-22	-34
larg. bande (Hz)	40	80	100	120	120

**Tableau D.15. haute-contre « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	370	630	2750	3000	3400
amp (dB)	0	-20	-23	-30	-34
larg. bande (Hz)	40	60	100	120	120

**Tableau D.16. soprano « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	800	1150	2900	3900	4950
amp (dB)	0	-6	-32	-20	-50
larg. bande (Hz)	80	90	120	130	140

**Tableau D.17. soprano « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	2000	2800	3600	4950

Valeurs	f1	f2	f3	f4	f5
amp (dB)	0	-20	-15	-40	-56
larg. bande (Hz)	60	100	120	150	200

**Tableau D.18. soprano « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	270	2140	2950	3900	4950
amp (dB)	0	-12	-26	-26	-44
larg. bande (Hz)	60	90	100	120	120

**Tableau D.19. soprano « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	450	800	2830	3800	4950
amp (dB)	0	-11	-22	-22	-50
larg. bande (Hz)	40	80	100	120	120

**Tableau D.20. soprano « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	325	700	2700	3800	4950
amp (dB)	0	-16	-35	-40	-60
larg. bande (Hz)	50	60	170	180	200

**Tableau D.21. ténor « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	650	1080	2650	2900	3250
amp (dB)	0	-6	-7	-8	-22
larg. bande (Hz)	80	90	120	130	140

**Tableau D.22. ténor « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1700	2600	3200	3580
amp (dB)	0	-14	-12	-14	-20
larg. bande (Hz)	70	80	100	120	120



**Tableau D.23. ténor « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	290	1870	2800	3250	3540
amp (dB)	0	-15	-18	-20	-30
larg. bande (Hz)	40	90	100	120	120

**Tableau D.24. ténor « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	800	2600	2800	3000
amp (dB)	0	-10	-12	-12	-26
larg. bande (Hz)	70	80	100	130	135

**Tableau D.25. ténor « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	600	2700	2900	3300
amp (dB)	0	-20	-17	-14	-26
larg. bande (Hz)	40	60	100	120	120

---

# Annexe E. Rapports de Fréquence Modale

## Contribution de Scott Lindroth

John Bower, un étudiant de Scott Lindroth, a dressé cette liste de fréquences modales pour différents objets et matériaux. Certains modes fonctionnent mieux que d'autres, et la plupart ne donnent des résultats plausibles que dans un intervalle de fréquences particulier. Caveat emptor.

En général, les objets en bois ne sonneront pas "bois" à moins qu'un composant aléatoire ne soit présent dans le son (essayez les guides d'onde en bandes). Néanmoins, certains des objets en bois font aussi de merveilleux instruments métalliques.

Ces rapports peuvent être utiles avec des opcodes comme *mode* ou *streson*.

**Tableau E.1. Rapports de Fréquence Modale**

Instrument	Rapports de Fréquence Modale
Dahina (tabla)	[1, 2.89, 4.95, 6.99, 8.01, 9.02]
Bayan (tabla)	[1, 2.0, 3.01, 4.01, 4.69, 5.63]
Plaque en bois de Cèdre Rouge	[1, 1.47, 2.09, 2.56]
Plaque en bois de Séquoia	[1, 1.47, 2.11, 2.57]
Plaque en bois de Sapin de Douglas	[1, 1.42, 2.11, 2.47]
Barre uniforme en bois	[1, 2.572, 4.644, 6.984, 9.723, 12]
Barre uniforme en aluminum	[1, 2.756, 5.423, 8.988, 13.448, 18.680]
Xylophone	[1, 3.932, 9.538, 16.688, 24.566, 31.147]
Vibraphone 1	[1, 3.984, 10.668, 17.979, 23.679, 33.642]
Vibraphone 2	[1, 3.997, 9.469, 15.566, 20.863, 29.440]
Plaques de Chladni	([62, 107, 360, 460, 863] Hz +-2Hz) [1, 1.72581, 5.80645, 7.41935, 13.91935] rapports
Bol tibétain (180mm)	( [221, 614, 1145, 1804, 2577, 3456, 4419] Hz) 934g, 180mm [1, 2.77828, 5.18099, 8.16289, 11.66063, 15.63801, 19.99] rapports
Bol tibétain (152 mm)	([314, 836, 1519, 2360, 3341, 4462, 5696] Hz) 563g, 152mm [1, 2.66242, 4.83757, 7.51592, 10.64012, 14.21019, 18.14027] rapports
Bol tibétain (140 mm)	([528, 1460, 2704, 4122, 5694] Hz) 557g, 140mm [1, 2.76515, 5.12121, 7.80681, 10.78409] rapports

Instrument	Rapports de Fréquence Modale
Ver de vin	[1, 2.32, 4.25, 6.63, 9.38]
Petite cloche à main	<p>([1312.0, 1314.5, 2353.3, 2362.9, 3306.5, 3309.4, 3923.8, 3928.2, 4966.6, 4993.7, 5994.4, 6003.0, 6598.9, 6619.7, 7971.7, 7753.2, 8413.1, 8453.3, 9292.4, 9305.2, 9602.3, 9912.4] Hz)</p> <p>[ 1, 1.0019054878049, 1.7936737804878, 1.8009908536585, 2.5201981707317, 2.5224085365854, 2.9907012195122, 2.9940548780488, 3.7855182926829, 3.8061737804878, 4.5689024390244, 4.5754573170732, 5.0296493902439, 5.0455030487805, 6.0759908536585, 5.9094512195122, 6.4124237804878, 6.4430640243902, 7.0826219512195, 7.0923780487805, 7.3188262195122, 7.5551829268293 ] rapports</p>
Sphère en spinelle de diamètre 3.6675mm	<p>([977.25, 1003.16, 1390.13, 1414.93, 1432.84, 1465.34, 1748.48, 1834.20, 1919.90, 1933.64, 1987.20, 2096.48, 2107.10, 2202.08, 2238.40, 2280.10, 0 /*2290.53 calculated*/, 2400.88, 2435.85, 2507.80, 2546.30, 2608.55, 2652.35, 2691.70, 2708.00] Hz)</p> <p>[ 1, 1.026513174725, 1.4224916858532, 1.4478690202098, 1.4661959580455, 1.499452545408, 1.7891839345101, 1.8768994627782, 1.9645945254541, 1.9786543873113, 2.0334612432847, 2.1452852391916, 2.1561524686621, 2.2533435661294, 2.2905090816065, 2.3331798413917, 0, 2.4567715528268, 2.4925556408289, 2.5661806088514, 2.6055768738808, 2.6692760296751, 2.7140956766436, 2.7543617293425, 2.7710411870043 ] rapports</p>
Couvercle de pot	[ 1, 3.2, 6.23, 6.27, 9.92, 14.15] rapports

---

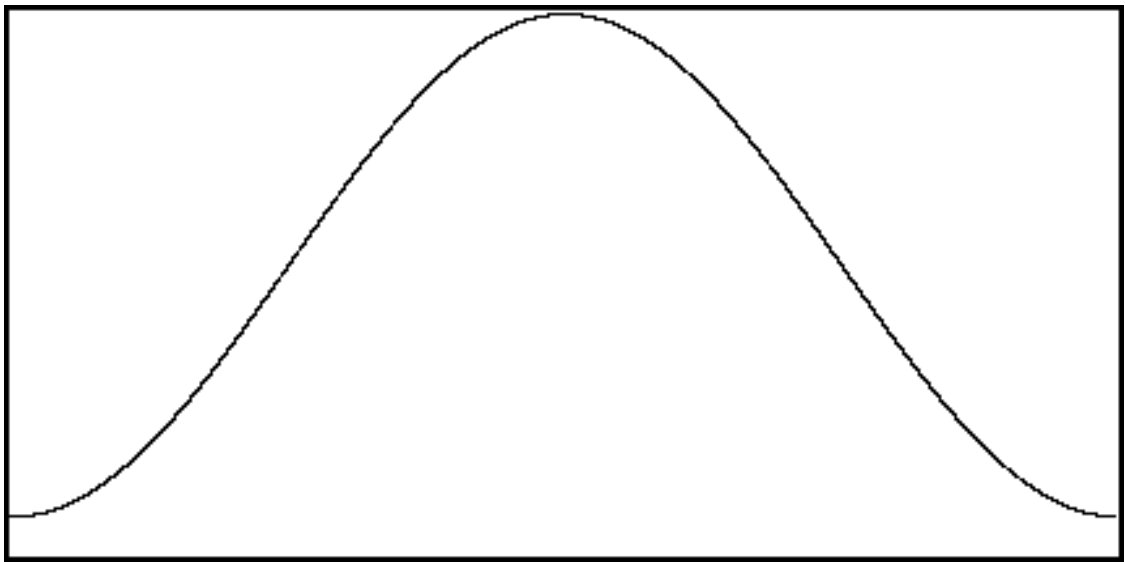
# Annexe F. Fonctions Fenêtres

Les fonctions fenêtres sont utilisées pour l'analyse, et comme enveloppes de forme d'onde, particulièrement dans la synthèse granulaire. Les fonctions fenêtre sont intégrées à certains opcodes, mais d'autres opcodes nécessitent une table de fonction pour générer la fenêtre. *GEN20* est utilisé à cet effet. Le diagramme de chaque fenêtre ci-dessous est accompagné de l'instruction *f* utilisée pour la générer.

**Hamming.**

## Exemple F.1. Instruction pour la fonction fenêtre de Hamming

```
f81 0 8192 20 1 1
```

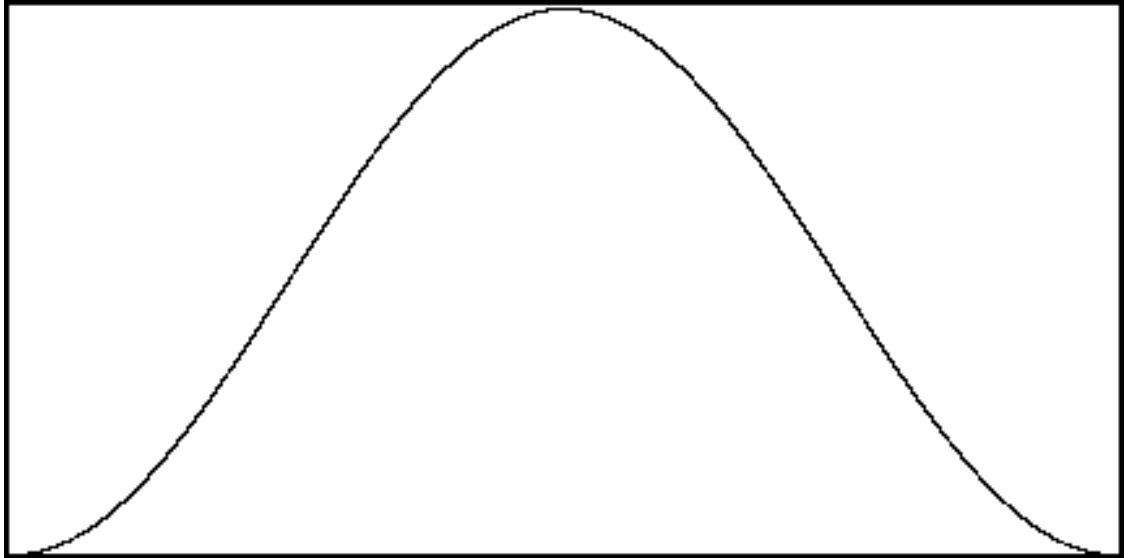


Fonction Fenêtre de Hamming.

**Hanning.**

## Exemple F.2. Instruction pour la fonction fenêtre de Hanning

```
f82 0 8192 20 2 1
```

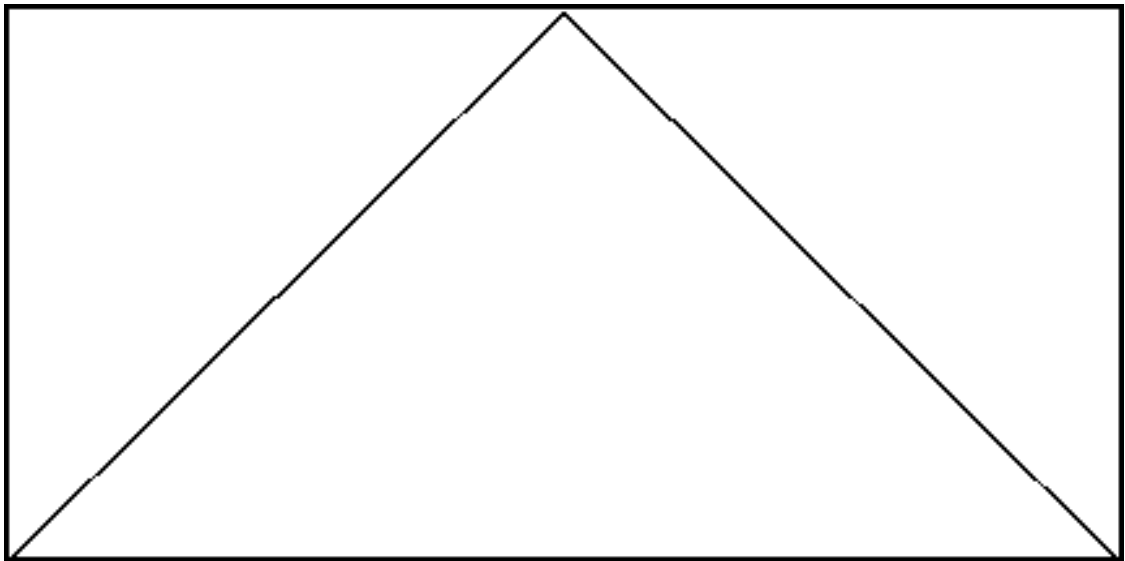


Fonction Fenêtre de Hanning

**Bartlett.**

### Exemple F.3. Instruction pour la fonction fenêtre de Bartlett

```
f83  0  8192  20  3  1
```

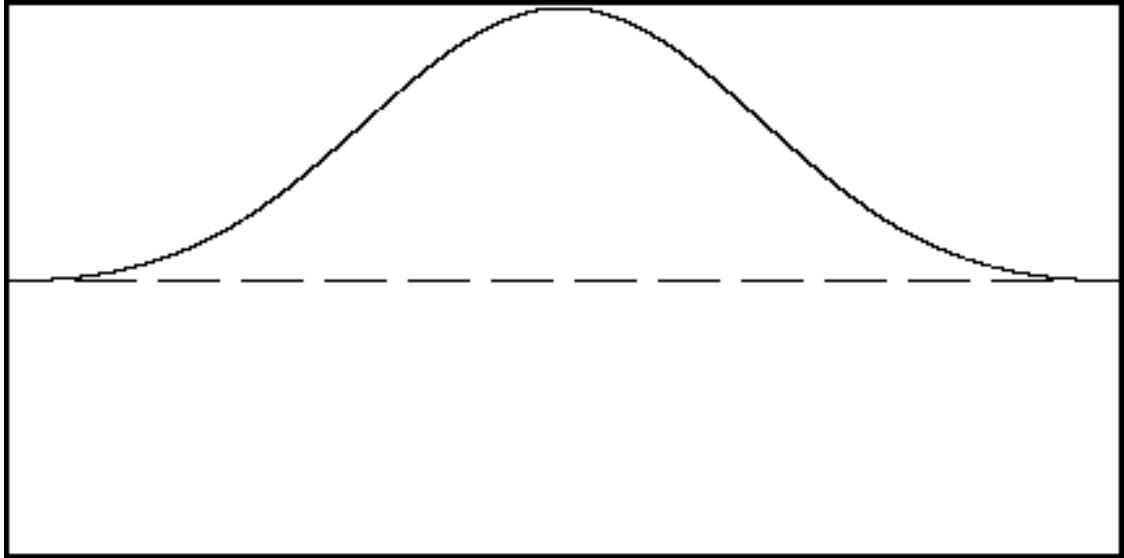


Fonction Fenêtre de Bartlett

**Blackman.**

### Exemple F.4. Instruction pour la fonction fenêtre de Blackman

```
f84  0  8192  20  4  1
```

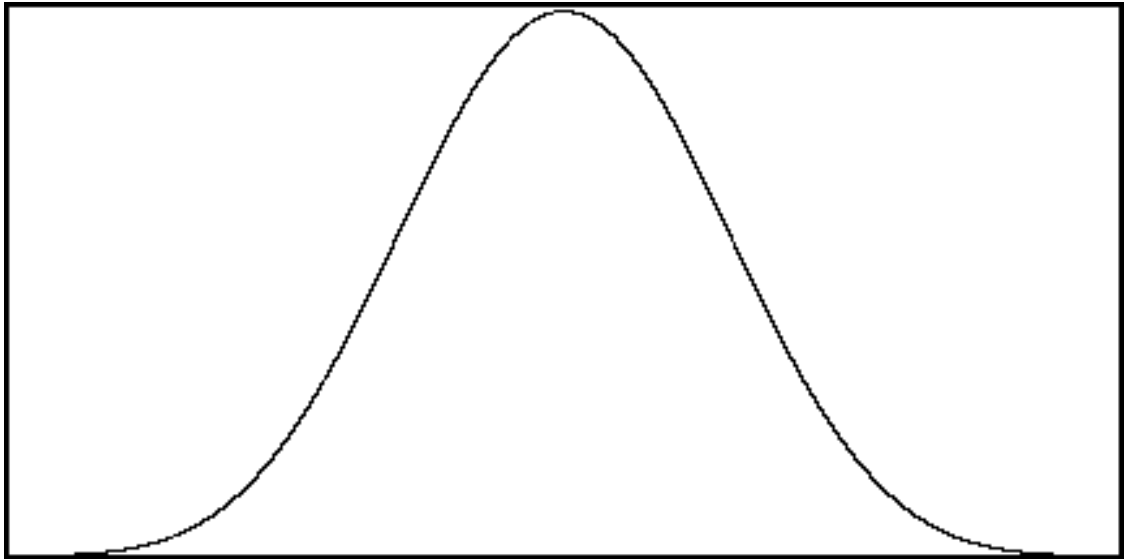


Fonction Fenêtre de Blackman

**Blackman-Harris.**

**Exemple F.5. Instruction pour la fonction fenêtre de Blackman-Harris**

f85 0 8192 20 5 1

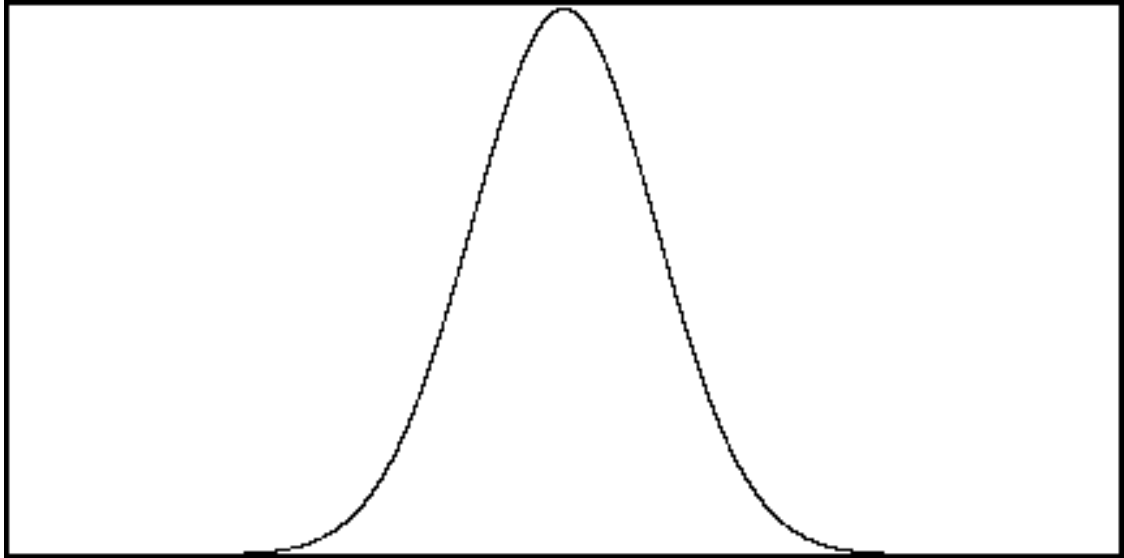


Fonction Fenêtre de Blackman-Harris

**Gaussienne.**

**Exemple F.6. Instruction pour la fonction fenêtre Gaussienne**

f86 0 8192 20 6 1

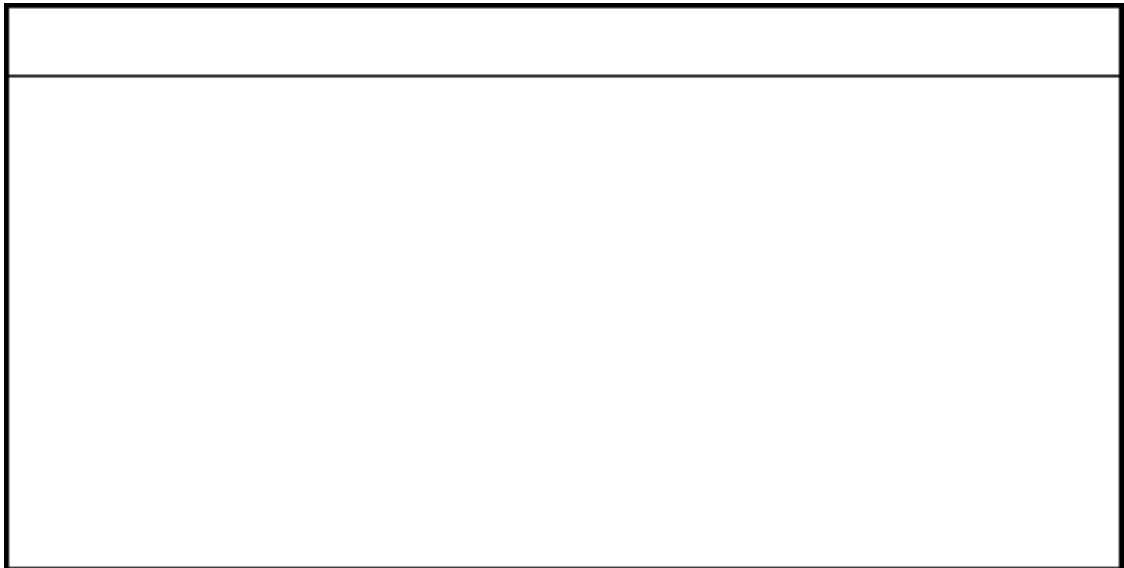


Fonction Fenêtre Gaussienne

**Rectangle.****Exemple F.7. Instruction pour la fonction fenêtre Rectangle**

```
f88  0  8192  -20  8  .1
```

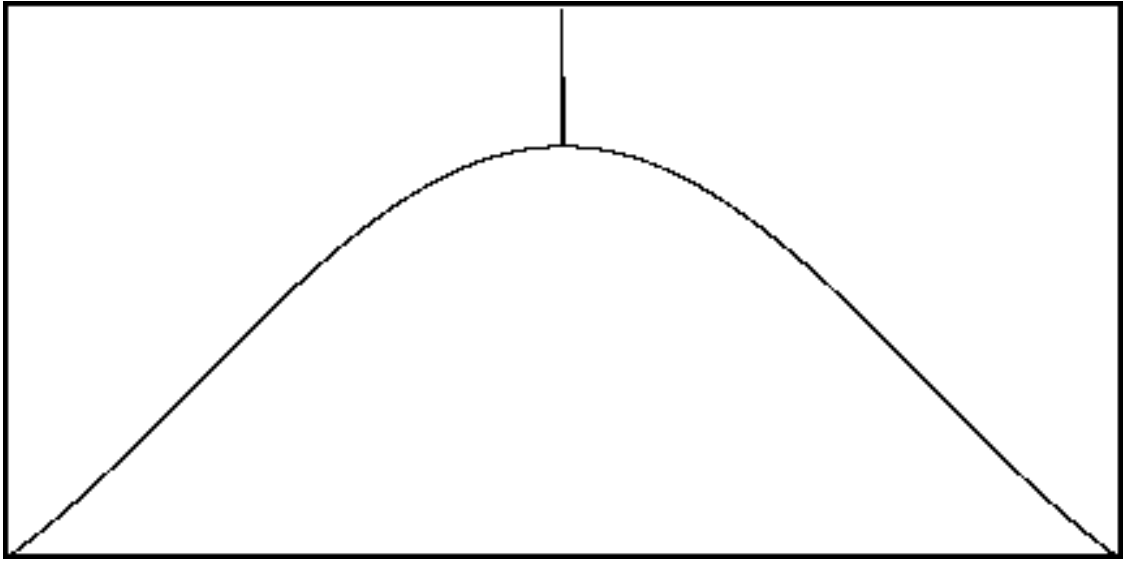
*Note* : l'échelle verticale est exagérée dans ce diagramme.



Fonction Fenêtre Rectangle

**Sync.****Exemple F.8. Instruction pour la fonction fenêtre Sync**

f89 0 4096 -20 9 .75



Fonction Fenêtre Sync



---

# Annexe G. Format de Fichier SoundFont2

A partir de la version 4.07 de Csound, *Csound* supporte le format de fichier de sons échantillonnés *SoundFont2*. SoundFont2 (ou SF2) est un standard répandu qui permet l'encodage de banques de sons basés sur des tables d'onde dans un fichier binaire. Afin de comprendre l'usage de ces opcodes, l'utilisateur doit avoir une certaine connaissance du format SF2, c'est pourquoi une brève description de ce format suit.

Le format SF2 comprend des objets générateurs et modulateurs. Tous les opcodes actuels de Csound concernant SF2 ne supportent que la fonction générateur.

Il y a plusieurs niveaux de générateurs ayant une structure hiérarchique. Le type de générateur le plus élémentaire est le « sample » (son échantillonné). Les samples peuvent être bouclés ou non, et sont associés avec un numéro de note MIDI, appelé la touche de base. Quand un sample est associé à un intervalle de numéros de notes MIDI, un intervalle de vélocités, une transposition (accord grossier et fin), un accord d'échelle, un facteur de pondération de niveau, le sample et ses associations constituent un « split » (division). Un ensemble de splits, avec un nom, constituent un « instrument ». Quand un instrument est associé avec un intervalle de touches, un intervalle de vélocités, un facteur de pondération de niveau, et une transposition, l'instrument et ses associations constituent un « layer » (couche). Un ensemble de layers, avec un nom, constituent un « preset ». Les presets sont normalement les structures de génération sonore finales prêtes pour l'utilisateur. Ils génèrent le son selon les réglages de leurs composants des niveaux inférieurs.

Les données des sons échantillonnés et les données de structure sont incorporées dans le même fichier binaire SF2. Un fichier SF2 unique peut contenir au maximum 128 banques de 128 programmes de preset, soit un total de 16384 presets dans un fichier SF2. Le nombre maximum de layers, instruments, splits et samples n'est probablement limité que par la mémoire de l'ordinateur.

---

# Annexe H. Csound Double (64 bit) ou Float (32 bit)

On peut construire Csound pour utiliser des nombres en virgule flottante DOUBLES sur 64 bit pour le traitement en interne au lieu des habituels nombres en virgule flottante FLOATS sur 32 bit. Cette plus grande précision pour le traitement interne produit un son bien plus "propre" mais au prix d'un temps de traitement plus long. Parce que csound met bien plus de temps pour ses calculs s'il a été compilé pour des doubles, il est utilisé typiquement en fin de travail pour produire la version finale d'une oeuvre. Si vous utilisez csound pour une sortie en temps réel, il vaut mieux utiliser une version 32 bit (float), qui fournit une sortie plus rapidement. Pour un rendu différé, vous pouvez utiliser l'une ou l'autre version, mais pour le master final, la version 64 bit produira une sortie de meilleure qualité.

La résolution choisie doit être la même que celle du type de variable des échantillons audio. Dans Csound "float" il s'agit de nombres en virgule flottante simple précision sur 32 bit. La mantisse occupe 24 bit. Dans Csound "double", il s'agit de nombres en virgule flottante double précision sur 64 bit. La mantisse occupe 52 bit. Chaque chiffre décimal nécessite 3 ou 4 bit. Ainsi, il y a une précision de 7 chiffres en "float" et de 16 chiffres en "double".

Pour chaque multiplication ou division, selon que les opérandes sont entiers, rationnels, décimaux périodiques ou irrationnels, le résultat peut présenter une erreur d'arrondi. S'il y a une erreur d'arrondi, il y a au plus une perte de précision d'un bit par opération (en plus des erreurs d'arrondi dues à la représentation binaire des nombres rationnels ou réels).

Pour les échantillons de type float, si le signal ne déborde pas la mantisse, le rapport signal-bruit vaut 6,02 fois 24, soit 144 dB. Au pire, chaque opération ajoutera 6,02 dB de bruit dû à l'erreur d'arrondi. Notre audition réagit à un ambitus dynamique effectif de 120 à 130 dB, mais nous apprécions que notre musique soit compressée dans un intervalle dynamique d'AU PLUS 60 dB (et habituellement beaucoup moins, disons 20 dB). Ceci nous donne  $(144 - 60) / 6,02 =$  environ 10 opérations défavorables avant que nous puissions entendre une dégradation. En pratique, nous pouvons enchaîner plusieurs fois ce nombre d'opérations avant d'entendre une dégradation ou du bruit.

Pour les échantillons de type double, si le signal ne déborde pas la mantisse, le rapport signal-bruit vaut 6,02 fois 52, soit 313 dB. Au pire, chaque opération ajoutera 6,02 dB de bruit dû à l'erreur d'arrondi. Ceci nous donne  $(313 - 60) / 6,02 =$  environ 42, en pratique plusieurs fois ce nombre d'opérations avant qu'il n'y ait une dégradation audible ou du bruit.

Mais si l'on relève le nombre d'opérations arithmétiques dans des instruments typiques de Csound ou d'autres synthétiseurs logiciels, les instruments très complexes entrent définitivement dans la zone des dégradations audibles sur de bons haut-parleurs avec float, et il n'est donc pas surprenant que dans certains cas, un test ABX à l'aveugle confirme des différences audibles *occasionnelles* entre de la musique synthétisée avec Csound "float" et la même musique synthétisée avec Csound "double". De même, il est courant de constater des différences audibles entre les implémentations numérique et analogique des mêmes algorithmes de synthèse.

Avec Csound double "l'espace de traitement numérique du signal" est nettement plus large, ce qui le rend plus adéquat pour toute musique dont l'écoute est critique. La version float ne devrait être utilisée que si son avantage en vitesse (environ 15 %) est critique pour l'exécution en temps-réel.

## Notes sur l'utilisation de Csound construit pour la double précision

1. Les fichiers *hetro*, d'analyse PVOC-EX et *pvanal* générés pour Csound 32 bit (float) fonctionneront

avec Csound 64 bit (double précision).

2. Les fichiers *lpanal* et *cvanal* générés pour Csound ne fonctionneront pas avec Csound64.

---

# Glossaire

## G

### Point de Garde

Un point de garde est la dernière position d'une table de fonction. Si la longueur est, disons 1024, la table aura 1024+1 (1025) points : le point supplémentaire est le point de garde.

Dans tous les cas, pour une table de 1024 points, le premier point aura l'index 0 et le dernier l'index 1023 (l'index 1024 n'est pas réellement utilisé).

Il y a un point de garde car certains opcodes lisent les valeurs de la table par interpolation ; dans ce cas, si l'index de lecture est par exemple 1023,5, nous aurons besoin de la position 1024 pour l'interpolation.

Il y a deux manières de remplir ce point (écrire sa valeur) :

1. La manière par défaut : en copiant la valeur du 1er point de la table
2. Le point de garde étendu : en prolongeant le contour de la table (en continuant le calcul pour un point supplémentaire)

En général le premier mode est utilisé pour les applications cycliques, comme un oscillateur (qui lit la table en boucle continue). Le second usage est pour les lectures à passage unique, comme les enveloppes, où il faut interpoler le dernier point correctement en suivant le contour de la table (on ne boucle pas sur le début de la table).