Giorgio Zucco

Inside Csound

Giancarlo Zedde
Giorgio Zucco, *Inside Csound*

Cover picture: Eduard Arancio  
eduardarancioweb designer@gmail.it  
English translation by Amedeo Casella  
©2014 Giancarlo Zedde  
Giancarlo Zedde  
Via Duchessa Iolanda 12, 10138 Torino  
www.zedde.com

*Csound* is an open source software  
Copyright 1986, 1992 by the Massachusetts Institute of Technology. All rights reserved. Developed by Barry L. Vercoe at the Experimental Music Studio, Media Laboratory, M.I.T., Cambridge, Massachusetts, with partial support from the System Development Foundation and from National Science Foundation Grant.

GZ0126
I dream of instruments obedient to my thought and which with their contribution of a whole new world of unsuspected sounds, will lend themselves to the exigencies of my inner rhythm.

Edgar Varese, 1937
Index

Foreword 11
To whom is this book addressed to? 13
Preface 15
Installation 18

Chapter 1 Programming language overview
1.1 Csound modules, opcode definition 22
1.2 Integrated environments, editor 23
1.3 A brief overview of WinXound 24
1.4 Hello world 27
1.5 Programming elements 33

Chapter 2 First steps, orchestra, score
2.1 Amplitude, frequency 38
2.2 0dbfs 39
2.3 Fundamental waveforms 41
2.4 Disk rendering 45
2.5 P-fields 46
2.6 Methods for events management 48

Chapter 3 Main techniques
3.1 Signal generators: introduction 54
3.2 Envelopes, glissandi 57
3.3 Conversions 61
3.4 Vibrato and tremolo techniques 68
3.5 Spatialization with Csound 71
3.6 Macro use 76
3.7 Randomness 77
3.8 Generative processes 80
3.9 Gen: overview 92

Chapter 4 Csound: MIDI and Osc
4.1 Introduction to MIDI-opcodes 98
4.2 Virtual MIDI keyboard 99
4.3 A first MIDI synth 100
4.4 MIDI controller 104
4.5 Read MIDI files 107
4.6 Microtonal scales 109
4.7 Complex events generation 111
4.8 Csound and OSC 116

Chapter 5 Graphical User Interfaces (Fltk, Max/Msp)
5.1 The Fltk Library 122
5.2 Fltk panel 123
5.3 Fltabs 125
5.4 knobs, sliders, buttons, counter, Joysticks 126
5.5 Automations, MIDI control 133
5.6 Snapshots 137
5.7 Graphics library with QuteCsound 139
5.8 Csound~ and Max/Msp 142
5.9 CsoundForLive 147
5.10 Cabbage and Vst development 150

Chapter 6 Additive and subtractive synthesis
6.1 Sound synthesis: introduction 158
6.2 Additive synthesis 158
6.3 Buzz oscillator 163
6.4 Modal additive synthesis 164
6.5 Subtractive synthesis, filters overview 171
6.6 Statevar 177
6.7 Moog filter 181
6.8 Formants filter 186
6.9 Equalizer 191

Chapter 7 Modulation techniques
7.1 Ring modulator 198
7.2 AM synthesis (amplitude modulation) 200
7.3 FM synthesis (frequency modulation) 201
7.4 PM Synthesis (phase modulation) 210
7.5 Waveshape 218

Chapter 8 Sample Playback
8.1 loscil 222
8.2 diskin2 225
8.3 Loop player 227
8.4 SoundFont 235
Chapter 9 Granular synthesis

9.1 Basic grain 240
9.2 Midi granular 243
9.3 grain2 244
9.4 grain3 246
9.5 syncgrain, syncloop 248
9.6 partikkel granular synthesizer 253
9.7 Vocal synthesis 263

Chapter 10 Physical models

10.1 Waveguide 270
10.2 The resonator 273
10.3 STK library 276
10.4 Modal synthesis 277
10.5 Scanned synthesis 282

Chapter 11 Spectral Processing

11.1 Phase vocoder 294
11.2 Time stretching, pitch shifting 296
11.3 Morphing 297
11.4 Harmonizer 300
11.5 Spectral blur 302
11.6 Spectral filter 303
11.7 Spectral delay 305

Chapter 12 Signal processors

12.1 Delay Lines 310
12.2 Reverbs 312
12.3 Chorus 317
12.4 Flanger 319
12.5 Phaser 321
12.6 Routing 322
12.7 Live electronics 326

Chapter 13U.D.O opcodes

13.1 Hipersaw U.D.O. 340
13.2 U.D.O. Chorus 343
13.3 Chowning, Risset Libraries 344
13.4 Vocal synthesis library 346
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>New syntax in Csound6</td>
<td>354</td>
</tr>
<tr>
<td>Appendix</td>
<td>357</td>
</tr>
<tr>
<td>Csound opcodes list</td>
<td>357</td>
</tr>
<tr>
<td>Csound in the web</td>
<td>366</td>
</tr>
<tr>
<td>Index of technical terms</td>
<td>367</td>
</tr>
<tr>
<td>Thanks</td>
<td>373</td>
</tr>
</tbody>
</table>
Foreword
by Richard Boulanger

Without compare, Giorgio Zucco is one of the finest Csounders in the entire world. He is a supreme master of the highest degree. He is a true “sound designer” - a title and skill that is in fact, quite rare in this gigantic “music industry” filled were “samplers”, and “remixers”, and “mashup-artists”.

Giorgio Zucco has “golden ears” and over the years he has produced an amazing “collection” of exquisite Csound instruments to prove it. I am quite confident that he could tame any machine and make her sing the sweetest lovesongs; but what he has done, is to dedicate his life to the creation of some of the most beautiful, elegant, powerful, engaging, evolving, mysterious, rawcous, raw, intense and tender musical timbres ever to be created in Csound.

Csound is deep and rich and arguably the world’s most powerful software synthesizer and signal processor.

And Csound is free... but there is a price. You have to “know” what you are doing. You have to know how sound “works”; you have to know how synthesis algorithms “work”; you have to know how DSP algorithms “work”; you have to know how psychoacoustics of music and acoustics and perception “works”. Giorgio Zucco has mastered all of these disciplines and translated that mastery into this incredibly important text - a book that opens his huge treasure chest of Csound “riches” to the next generation of electronic and computer musicians.

I have learned so much from Giorgio’s work, and I have been a huge fan of his for years. In fact, I have showcased his instruments in all of my classes and featured his instruments in all of my concerts. And I am not the only one, my dear friend and mentor, Max Mathews, the father of computer music, was a huge fan of Giorgio Zucco too.Max especially loved and used Giorgio’s Scanned Synthesis and Pierce Scale instruments! We both did and I still do (Max’s 80th Birthday 2007).

Csound today is experiencing a Renaissance. Today, the Csound engine and code is running as an “external” in Max/MSP and PD; as an “instrument” and “effect” in Ableton Live; as a softsynth and signal processor “app” on the Android and the Apple iPad; and as an “Audio Unit” and “VST” plugin in virtually every DAW on the market! Csound has been born again, and the dawn of the Italian Renaissance of Computer Music is heralded by Giorgio Zucco and his new book. Giorgio’s Sintesi digitale del suono will light the way and inspire for years to come.
I look forward to it’s translation into many languages, and to a long life in the minds and music of the next generation of innovators that he will inspire through it. Giorgio Zucco is a rare sonic master who, through this brilliant and inspiring book, has shared many of his hard-earned secrets and extraordinary audio treasures with the world.

Dr. Richard Boulanger, Ph.D.

Professor of Electronic Production and Design
Professional Writing and Music Technology Division
Berklee College of Music
1140 Boylston Street
Boston, MA 02215-3693
To whom is this book addressed to?

*Inside Csound* is mainly aimed to all the musicians who love music and technology. It is not a purely theoretical text, but it’s a practical programming manual suited either to the aspiring sound designer or the musician with an academic background. The book offers a step by step exploration of what is truly a sound factory. Concepts and techniques presented in this book have, perhaps, the ambition to start always from a practical application in a real music context and not only technical. Although *Csound* found through the years many applications also in the field of scientific research, I believe that its nature – as its origins – is that of applications in the music and art field. *Csound* it’s a tool to make music, we could even consider it a true and real musical instrument, a specific instrument to make computer music, namely the extraordinary alchemy that fuses music composition and informatics.

The fundamental idea that brought to the birth of this text is linked to the history of *Csound*. It is more than ten years since the publication of two texts that I consider the Bible of *Csound*: *Csound book* (Boulanger, MIT Press, 2000) and *Il suono virtuale* (Bianchini, Cipriani, ConTempoNet). The present text doesn’t want to take the place of what is the seminal literature of computer music, however tries to focalize on a fundamental aspect, specifically informatics that shouldn’t be underestimate.

For many years, too often *Csound* was thought as a tool only suited to prerecorded synthesis, a point of view that, if today doesn’t appear as wrong, is at least mean for several reasons.

As for many programming languages, *Csound* too evolved during the years, not only it increased extraordinarily the number of opcodes, but many creative ways were opened, new programming patterns, new esthetic and technical goals. There are very many musicians, sound designers, simple amateurs, graduates in Electronics at the Conservatoire of Music, who probably don’t know that many of their instruments, maybe written ten years ago, today could be rewritten and optimized with less line codes, could be converted in tools for the real time, could become plugins for audio/Midi sequencers, or could be played on their own tablet!

This is the reason which a new text in *Csound* could today find the right place: the intention to tell by many practical examples the incredible potential and novelty of what is likely the most important and powerful language for synthesis of sound. Nowadays we can talk of a new real and true renaissance, as Richard Boulanger says.

How to approach the text? Is it fundamental to follow step by step starting from the first chapter? For beginners the answer is of course affirmative. However, also for the musicians that come from a more traditional academic background it’s advisable to read and try all the material up to the fourth chapter, this, in order not to miss some
ways of programming considered as standard for this language.

The book is divided in two wide sections.

• Section one: from chapter 1 to chapter 5 we find all that belongs to basic syntax, from the use of editors in the process of compilation, to the introduction of Midi opcodes and the construction of gui. Chapter 3, in particular is the widest and offers a vast panoramic of the main programming techniques in Csound, control and generation of events.

• Section two: from chapter 6 to chapter 13 we tackle diverse techniques of synthesis from the oldest to the newest. The chapters have many examples that show the construction of different techniques using control in real time, graphic interfaces and music applications.

The author is grateful for any suggestion sent to the e-mail address: giorgiozucco@teletu.it

Files in .cds format of each chapter can be requested free of charge to the Editor: www. zedde.com
Preface

We are in the pocket recording studios era... in which it’s possible to emulate (with stunning realism and quality), machines such as Neve, SSL, Trident, Harrison banks, easily on our laptops. The modern convolution plugins, or physical models, can reach the warmth and features of a Distressor compressor to one-twentieth of the original machine cost.

And speaking about sampling? The oboe sample with his 100 kb size, running on the glorious 12 bit Akai S900, today has his own graphical interface, it’s 24 bit sampled in every possible articulation, and with three different microphones positions.

And what about synthesis? The power of modern CPUs allows music software houses to produce increasingly sophisticated software synthesizers, here below is the presentation of a famous Vst (no name is given):

- Semi-modular synthesizer offering an endless amount of sound possibilities
- New combinations of multiple synthesis and sampling techniques
- Amazing improvement implemented in the new interface
- 1200 presets
- Amazing effects, and wave morphing control
- 16 “macro control” freely assignable and a new section for the ADSR modulation, to have an easy and complete control of sound
- Chance to divide the envelope into a maximum of 68 breakpoint
- Advanced function for surround managing for each channel

...and then:

Ability to open endless instances of the plugin. With a modern quad core CPU it’s possible to easily handle 20-25 instances of a very complex plugin, all in real time of course. Despite such prodigies, detractors of synthesis languages ask always the same question: “It really makes sense dedicating to a programming language now “old”, which requires computer skills, working in deferred time with a long learning curve, in which to create a 4 seconds sine wave, several minutes of code writing, debugging, compiling and final render are needed?”

Predictable answer ... but let’s start with making the necessary clarifications regarding
some inaccuracies that circulate around this synthesis language.

To be a computer expert is not mandatory to learn Csound! Its syntax, overcome the natural difficulties of the starting approach, it’s relatively simple. Building a Csound instrument requires a mindset closer to the concept of modular synthesis, each Csound object (not quite all...) has an input and an output.

Who experienced with traditional programming languages such as Java, for example, certainly recalls complex concepts such as classes, inheritance and polymorphism, methods, and much more. You will not find anything like this in programming Csound (for the moment at least).

Csound allows real time operations from every point of view, you can build your own synthesizer and play it in real time with your master keyboard, checking every parameter of the sound using knobs, sliders, ribbon controller of your own MIDI controller. Csound can also process external signals in realtime. Despite in this subject, current standards are still Max / MSP, Pd and Super Collider, it’s a field with significant potential still to be explored.

Csound offers a perfect timing and low latency for live performance, thanks to the ability to interface with any audio converter using external drivers such as ASIO.

Csound is available for all platforms (Mac, Linux, pc, tablet).

From the educational point of view, in addition to official publications, there is a very extensive documentation on the web as well as several libraries, compositions, tutorials for different techniques.

Recent Csound versions offer the opportunity to build graphical user interfaces, reaching a level of interactivity very similar in Max / MSP, the graphics library used is called Fltk (Fast Light Toolkit, mostly used on Linux platforms). These opcodes were introduced for the first time by the researcher musician Gabriel Maldonado in the CsoundAV version (which, among other things, also incorporated OpenGL 3D graphics). However, recent improvements of this language are gradually bringing developers to prefer other methods of graphics construction, such as integration with Max / MSP, Pure Data and the latest Cabbage.

Csound does not get old... it doesn’t follow vogues... It’s not born with specific musical purposes. It’s an instrument suitable for both the electro-acoustic composer of classical nature, and the Djs - examples of applications in genres as electronic house, techno, minimal, industrial, glitch, are not rare.

Csound is maintained and updated by an international staff of the highest level, in this very moment the developers will probably be filling out a new release.

Csound can be used as synthesis engine for other languages such as Max / MSP and
Open Music, with these two graphical environments it’s possible to control Csound via MIDI, generate scores, do some algorithmic composition, develop plugins for Ableton Live, and more!

...I forgot... among this... it’s free...

Giorgio Zucco
Turin, June 2103
Installation

First of all let’s download the latest Csound version from http://csound.sourceforge.net/, and let’s choose the version suitable for our machine. Installation is very simple, the executable contain the complete package including the compiler, a lot of educational material with various example files (including the famous piece Trapped in Convert by Richard Boulanger that can be a fresh star to check the sound possibilities of this language), an editor (to write and compile Csound sources) called qutecsound, and other .dll files containing new opcodes.

Notes for the Windows version: in the installation package for Windows (version 5.1 to latest versions the problem could be solved the next release), there may be a small bug (generally on 64-bit systems) with the python25.dll file. During the synthesis process the Csound compiler may generate an error message indicating this file is missing. To solve this problem you need to install the Python programming environment from the site http://www.python.org/
If the problem persists, it’s better to manually copy the python25.dll file in the system32 folder.
Installation on Osx: unpack the .dmg file, launch the .pkg file and the compiler installation begins with his libraries. Finally copy qutecsound editor icon in the Applications folder.

Notes for qutecsound (Windows, Mac): it’s better to use an up-to-date qutecsound version (in latest versions it’s called CsoundQt) from Andrea Cabrera’s site:

http://sourceforge.net/projects/qutecsound/files/

There is also a distribution for Linux, but it’s not covered in this text.

Please note that for advanced users, it’s possible to download the Csound source package (it’s open source) and compile it using appropriate development tools. For more info, you should read the official manual:

Chapter 1
Programming language overview

1.1 Csound modules, opcode definition
1.2 Integrated environments, editor
1.3 A brief overview of WinXound
1.4 Hello world
1.5 Programming elements
1.1 Csound modules, opcode definition

How a Csound program is structured? Imagine a header with formal statements (a header is an opening *name* declaration for a precise task, a process *closure* instruction, and the behavior over time of a determinate process (for example the duration). To achieve this Csound uses particular english keywords, unit generators, opcodes (operation codes).

opcodes are code blocks written in C language, defining a precise function inside the program (mathematical calculations, generation of a simple sine wave or a complex synthesis technique, integration of a whole algorithm for a particular signal processing and so on). Imagine the procedure to bake a cake (!?!?)... , to know the type of ingredients we need and the way to combine them. To get the final result (yes... a cake), our last concern is to know how each ingredient is produced! To create a sound in Csound, the sound designer should know the theory of a synthesis technique, and he should use the right **opcodes** to get the sound.

The latest version of Csound is the 6.02, since the first version by Barry Vercoe (1985) a lot of changes has been introduced, in comparison to previous versions, more “deferred” (working with orchestra and score separated, with render on disk). Latest versions offer a variety of tools for real time, creation of graphical user interfaces, streaming of any type of audio file, perfect integration with any audio or MIDI and much more.

Csound 6.02 introduces the following features:

- PortAudio with ASIO driver support (which ensures low latency for real time).
- Portmidi to control Csound with any MIDI controller on the market.
- Fltk (fast light toolkit) graphics library for objects such as sliders, knobs, buttons, joysticks, etc..
- FluidSynth, module consists of various opcodes for file management in soundfont format.
- Python opcodes to initialize the Python language interpreter from an orchestra.
- OSC opcodes: OSC (Open Sound Control) is an electronic protocol alternative to MIDI that allows data exchange between electronic musical instruments or computers at very high speed, but currently not supported yet by the most of music hardware on the market.
- ATS, Loris, PVS: are new generation opcodes for the analysis and re-synthesis also in real time.
- STK opcodes (Perry Cook’s original Synthesis Toolkit in C++): open source library for physical modeling synthesis.
Chapter 1  Programming language overview

• CsoundAPI: Application Programming Interface, Csound API allow the use of this powerful synthesis engine inside different programming languages (Java, Lisp, C++, Python, Lua), one of the main exploration fields is Vst plug-in format programming.

1.2 Integrated environments, editor

The current version of Csound6 contains an excellent editor called qutecsound (now called Csoundqt) included in the installation package. Qutecsound offers all the typical features of a modern Csound editor and much more:

• Colored Syntax
• Tools to optimize and speedup analysis and re-synthesis techniques
• Widget creation (objects for graphical user interfaces)
• Instant access to the syntax of all the opcodes
• Tutorials for the main techniques

There are many other editors for this language, some of these have not been maintained or updated over the years (typical problem related to the world of open source softwares), while others have become standards for many csounders around the globe. The following list is only indicative and does not take into account historical editors no longer supported, it’s worth mentioning:

• Winxound by Stefano Bonetti: is my favorite editor, incredibly stable and effective from every point of view! available for all platforms.
• Blue by Steven Yi, is a system that recall the interface of a multi-track sequencer, offers several tools for composition and score control, available for all platforms.
• Cabel an experimental version similar to graphic systems as Max, Open Music, Pure Data, the latest version was released in 2006 and we have no information on further developments.
• Cabbage by Rory Walsh, it’s a tool for building standalone graphical environments and Vst. In the fifth chapter of the book we will talk about this extraordinary tool.
1.3 A brief overview of WinXound

A very good editor developed and continuously updated by Stefano Bonetti. Initially implemented only for Windows and now also available for Mac and Linux, is an optimized version to work with Csound5 (may be incompatible with older versions), and offers the following features:

- write and edit CSound, Python and Lua files (csd, orc, sco, py, lua) with colored syntax;
- working with Csound, CsoundAV (G. Maldonado version), CSoundAC (experimental version for algorithmic composition by Michael Gogins), Python and Lua compilers;
- allows interfacing with other editor (as QuteCsound);
- offers very practical tools for analysis and re-synthesis techniques, in which is
possible importing an audio file (wav, aiff) and creating Csound files for future re-synthesis works;

• integrates the official Csound manual which can be accessed immediately and directly during code writing;

• imports and automatically converts separate .orc and .sco files into a .csd file

• numbered lines, it’s very important when the compiler throws a markers syntax error;

• integrates a collection of Udo opcodes, not contained in the official version of Csound, not “compiled” but written directly in Csound language (the final chapter of the book will treat this programming technique).

From the menu – File – Settings it’s possible to configure the editor in great detail, for example, we can associate an external audio editing software (such as the open source Audacity or Ocenaudio), we can specify the CsoundAV directory for analysis and re-synthesis operations (Menu – Tools – Csound Analysis) and select a folder that contains our samples or where we want Csound writes analysis files.

Initial Configuration:

• Windows: From the menu – File – Settings – Compiler Settings, you will find the voice Csound Compiler Default Flags, these are the startup settings for Csound commands for file rendering, by default are:

-B4096 --displays --asciidisplay

It’s recommend to not modify it, the flag -B4096 is the hardware memory buffer; by using good external sound cards (firewire or usb) it’s possible to reduce latency for real time by lowering this value. It’s recommended to keep these settings using the built-in sound card.

• Mac: menu-WinXound-Preferences-Compiler, use as flags:

-b4000 -B4000 -g

Notes: the --asciidisplay flag could generate a crash with examples that use the Fltk library, in case of error remove this flag.

In later chapters we will see how to customize these command lines. One possible line to read an audio file in real time would be:

-odac1 -b4000 -B4000 -g

The number “1” of odac flag (or devaudio) activates a port on the sound card, it’s possible to specify a number between 0 and 1023. Pointing to a wrong number Csound throws an error and provides a list of audio devices scanning the computer, so it’s possible to find out the correct number. You can also write -o separated by dac, “-o dac”.

25
In most of the examples of the book, –odac flag will be used, followed by a number (audio port of the hardware device) indicated in the .csd file in this way:

```xml
<CsOptions>
  -odac1
</CsOptions>
```

-odac allows real time rendering, it means that we can listen to an instantly created file without waiting for the render on the hard disk. The –odac flag is a Csound default setting. If we did syntax errors, the compiler will warn us indicating the type of error:

```
new alloc for instr 1:
a1 oscili.kk kamp kfreg 10 0  
  B 0.000Score finished in csoundPerform().
inactive allocs returned to freespace
end of score. overall amps: 0.00000
  overall samples out of range: 0
1 errors in performance
Elapsed time at end of performance: real: 0.122s, CPU: 0.122s
0 128 sample blks of 64-bit floats written to dac
------- Compiler End -------
```

Compiler Info/Warnings/Errors:
INIT ERROR in instr 1: Invalid ftable no. 10.000000
  - note deleted. i1 had 1 init errors

During the exploration of the various synthesis techniques and audio processing, we will see many flags for disk rendering, to enable MIDI ports, etc.
For a complete list of command lines in Csound consult:

Notes
Csound versions after 5.14, may present some problems with examples that use the graphics library Fltk, in case of error messages (both on Mac and Windows) it can be necessary adding the following flags in the options:

```xml
<CsOptions>
  --old-parser -d
</CsOptions>
```

About the default flag in Winxound (on Windows):

```
-B4096 --displays --asciidisplay
```

I have noticed some problems related to the use of Fltk, in these cases it’s recommended to keep the setting:

```
-B4096
```
Chapter 1 Programming language overview

The use of modern editors as Csoundqt and Winxound, now represent a must in order to optimize the work of writing and compiling effectively. For more “adventurous” users it’s also possible to write the text file using a text editor (for example TextEdit on Mac) and start Csound compiler through Dos (on Windows) or Terminal (OSX).

An example of a compilation from the terminal on OSX:

```
$ /usr/local/bin/csound -d -A -v Test.csd -o /Users/user/ CsoundExample/Test.aif
```

As we see it’s not very intuitive and practical using the command line (where we can see the the compiler path, the files to run and the final audio file). The use of this system, however, becomes essential to control Csound from custom environments, for example, to interact with other languages with controlling purposes.

1.4 Hello world

*Hello World* is the simplest program to begin studying any programming language. A *Hello World* program in *Java*, for example, once compiled returns to the command prompt the message...*Hello world*.

Our *Hello world* with Csound will of course be a sound (although it’s also possible with Csound text printing on a window, for example, to display the result of a mathematical calculation, to monitor the signal level, etc.), which allows to begin understanding Csound key features.

On Windows or Mac, open the WinXound editor, as a new file format we choose the Csd (Unified File Format, introdotto nella versione di Csound 3.50; this format was introduced for the first time by Michael Gogins in AXCsound). The practicality of a file with the .csd extension is having the two constituent elements of a Csound program in a single file:

Notes

Modern integrated environments mentioned (as Winxound) are optimized for .csd files writing and reading, of course it’s possible to open and edit files or Csound examples made for previous versions, which will then be divided in two separate files (.orc and .sco). If we try to open an old file taken from the immense web archive, or from historical texts such as *Virtual Sound* (Bianchini-Bob) or *Csound book* (Boulanger), the program will open the two files directly in the modern .csd format. Structure of a Csound program:

**orchestra**: program block containing the synthesis algorithm (or signal processing, etc.). An orchestra can contain a single instrument (for example an emulation of a plucked string, a synthesizer for frequency modulation, a filtered white noise generator, an effect bank, etc.), or a multiple instruments set. Each Csound instrument is defined within the following syntax:
it’s also possible to indicate the instrument name with a string:

```plaintext
instr Risset_Bell
...algorithm description...
endin
```

we can combine several instruments inside the orchestra:

```plaintext
instr 1
...algorithm...
endin

instr 2
...algorithm...
endin

instr 3
...algorithm...
endin
```

Let’s see an example of Csound instrument, a simple oscillator:

```plaintext
instr 1 ;(or: instr oscillator)
kamp = .8
kfreq = 440
ifunc = 1
audio poscil kamp, kfreq, ifunc
out audio
endin
```

**score**: it’s a sort of electronic score in which the events are defined in time and they are relative to instruments written in an orchestra. A Csound score can be complex in accordance with the number of instrument control parameters, or it can be incredibly simple in case of a live performance, in that case the function of the score will be only related to the performance duration itself.

A score example with five control parameters:

```plaintext
;i1 p1 p2 p3 p4 p5 ;(csound ignore the code after
the “;”)
i1 0 1 .8 110
i1 1 1 .8 220
i1 2 1 .8 440
i1 3 1 .8 880
i1 4 1 .8 440
i1 5 1 .8 220
```
Chapter 1  Programming language overview

or using a word as instrument name:

```
i"guitar"	0	1	.8	110
i"guitar"	1	1	.8	220
i"guitar"	2	1	.8	440
i"guitar"	3	1	.8	880
i"guitar"	4	1	.8	440
i"guitar"	5	1	.8	220
```

a score can of course control several instruments at the same time (imagine an orchestral score):

```
i"guitar"	0	1	.8	110
i"bell"	1	1	.8	220
i"snare"	2	1	.8	440
i"guitar"	3	1	.8	880
i"noise"	4	1	.8	440
i"bell"	5	1	.8	220
```

and in case of a live performance? what about real time control of sound parameters? maybe with an external MIDI controller? it will be enough givin the live event duration only:

```
f    0    3600
```

this brief statement indicates that a given sound process will remain active for a hour. In the earliest versions of this language was necessary writing your own code on two separate files with the extension .orc (orchestra) and .sco (score). The .csd format, for its usability has finally become the standard since many years.

Let’s analyze the skeleton of a .csd file:

```
<CsoundSynthesizer>
<CsOptions>
    ;this space is dedicated to Csound flags, flags are commands with which you can ask the compiler to perform many tasks, such as: writing files to disk, compiling in real time, communicating with your audio or MIDI hardware and much more;
</CsOptions>
<CsInstruments>
    ;this space is called Orchestra, is the program block in which defining instruments, for example:
    instr 1    ;we created an intrument called “1”
    ;in this space we will define the instrument features
    endin    ;closure instruction for instrument “1”
    instr 2
    endin
```
instr 999   ;we can use any number or build several instruments
endin
</CsInstruments>
<CsScore>
;in this space we will write the Csound score, is a score to
define temporal events in connection with instruments created
in the orchestra, it will be possible to say, for example, how
many seconds a certain instrument will play, when it starts to
play in terms of seconds and much more.
</CsScore>
</CsoundSynthesizer>

You may have noticed the use of semicolons in file comments, so you can comment
the code of a csound file because the compiler will skip all that is written after the
semicolon. To do longer comments without having to constantly use “;” symbol we
can delimit the text within a /* .... */ block, for example:
/*
...here we can write a comment without length or lines limits...
*/

From the point of view of learning any programming language, it’s fairly common
and recommended practice to comment code lines. This serves both to the beginner
who took his first steps and the experienced programmer who want to put his hand
on previously written programs without forgetting the meaning of certain passages.
Come back to our program skeleton and insert the following code lines:

Esempio Cap1.1_hello world.csd

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
chnls = 1
instr 1
asound  rand 10000
out asound
endin
</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
Chapter 1 Programming language overview

Now save the just written file with WinXound, (File-Save as), under the voice Tools we find the Compile command, if we have not made any syntax errors Csound will start the real time compilation process, if all went well you should be able to hear 10 seconds of white noise.

Notes: during the file compilation Csound does not take into account letters spacing, a statement such as:

```
asound    rand    10000
```

needs a space between words because they are three different arguments (variable, opcode, opcode parameter), but it’s “irrelevant” the size of this space. Examples such as these who follow do not generate error and return the same result:

```
asound  rand  10000
asound   rand  10000
```

other examples of how to use spacing concern score writing, the following two snippets are identical:

```
i1 0 8 440
```

or:

```
i 1 0 8 440
```

Observe written code lines in detail, for first we see the Csound Header, it’s a very important part of programming in which we define for example the sample rate and the number of channels, however, in this simple example file “header” is not mandatory because it represents default values. If we were at lower control rate instead, an eight-channel audio system or a higher sample rate, header changes will be indispensable.

```
;Header
sr = 44100    ;44100 sample rate
kr = 4410     ;control rate
ksmps = 10    ;sr/kr ratio; sets the number of samples in a control period.
nchnls = 1    ;number of channels, mono in this case
instr 1       ;define an instrument “1”
asound rand  10000
```

In this code fragment we create a white noise generator with amplitude 10000, what is “asound”? is it a Csound command? absolutely not, “asound” is the audio type variable name that we called “sound”, from what we know that it’s an audio signal generator? just by the first letter “a”. In Csound there are different types of variables:

- “a”: all variables of audio type begin with that letter, after the letter “a” you can add any word or number, such as a1, a2, a77, a99, asound, anoise, etc.
“i”: are initialization variables, it will be discussed speaking of waveforms;

“k”: control variables, are variables that don’t generate audio but define behaviors of something like a sound generator. This may be a variable controlling the amplitude envelope of a given sound, or the one controlling the cutoff frequency of a filter.

“g”: global variables, must be declared in the orchestra before instruments construction, these variables can be accessed by all the instruments of the orchestra.

Now come back to our code fragment:

```plaintext
asound rand 10000
```

So we defined the variable “asound”, now move to `rand`, it’s an opcode that generates random numbers in a given range. By using this opcode as the audio signal we get white noise, often used in subtractive synthesis for its complex sound density;

```plaintext
out asound
```

```plaintext
endin
```

“out” is an opcode that connects the sound generated by csound to our sound device, there are many opcodes to control the output sound, out is the simplest (a single mono channel), csound also offers multichannel audio management, four channels setting, eight channels setting and 5.1 systems.

The final statement “endin” closes the program block called “instr 1” at this point we have to tell csound how to operate this just created sound:

```plaintext
<CsScore>
    ;p1  p2  p3
    i1   0   10
</CsScore>
</CsoundSynthesizer>
```

What are p1, p2, p3? They are Csound pfields, they indicate the parameters of a particular sound, a Csound instrument can have numerous pfields (imagine a sound score where we have to define attack, duration, amplitude, frequency, pan and much more), the basic parameters of a sound are the first three pfileds:

- p1: instrument name, such as “i1” will refer to the orchestra instrument “instr 1”
- p2: attack, which indicates when (in terms of seconds or milliseconds) this instrument will sound, indicating p2 = 0 Csound will play the sound at the instant zero;
- p3: duration, indicates the duration in seconds, the statement of our example:

```plaintext
i1   0   10
```

is asking the compiler to play “instr 1” (our white noise) for 10 seconds with instant attack.

We saw a first simple example of sound synthesis in Csound, try changing the file you just created in the following way:
• change the amplitude value of rand (be careful ... do not put values as impossible values such as dozens of zeros ...);
• change the duration of this sound
• try to create several instances of instr 1 with various time intervals.

1.5 Programming elements

In the short example “Hello world” we met some elements and basic concepts of programming:
• constant: fixed value that remains unchanged while the program is running
• variable: is used to hold data that vary over time during the execution of the program
• algorithm: description of the logic according to which a program processes data to perform a certain task.

How variables with numeric values can be processed? For example, with simple mathematical operations. In Csound there is a large opcodes list to do arithmetic operations. Let’s see a simple program where two variables are used for mathematical operations:

Example Cap1.2_mathematics

```csound
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1
ivarA    =  8    ;variables declaration
ivarB    =  4
iaddition =  ivarA+ivarB
isubtraction =  ivarA-ivarB
idivision =  ivarA/ivarB
imultiplication =  ivarA*ivarB
isinA     =  sin(ivarA)
icosA     =  cos(ivarA)
irnd      =  rnd(ivarA)
isqrt     =  sqrt(ivarB)
iespress  =  (((ivarA+ivarB)/2)*cos(ivarA*ivarB))/ivarB+sqrt(ivarA)

;prints to the console, print also accepts multiple arguments (print var1, var2, var3,etc...)
print iaddition
print isubtraction
print idivision
```

33
print irnd
print imultiplication
print isinA
print icosA
print isqrt
print iespress
endin

</CsInstruments>
<CsScore>
i1 0 20
</CsScore>
</CsoundSynthesizer>

program prints on Csound console the following output

SECTION 1:
new alloc for instr 1:
instr 1:  iaddition = 12.000
instr 1:  isubtraction = 4.000
instr 1:  idivision = 2.000
instr 1:  irnd = 7.788
instr 1:  imultiplication = 32.000
instr 1:  isinA = 0.989
instr 1:  icosA = -0.146
instr 1:  isqrt = 2.000
instr 1:  iespress = 4.080
------- Compiler End -------

Another typical aspect of a programming language is conditionals, substantially a condition is defined: if the condition is “True” the program executes a defined statement (or a process starts, etc.), if the condition is “False “the program ignore the statement and goes to the next line.

The block “if-then”:
if “declare conditions” then
…..statements
endif

Example Cap1.3_if-then

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
sr   = 44100
kr   = 4410
ksmps = 10
nchnls = 1
instr 1
  ivarA = 5 ;variables declaration
  ivarB = 4
  if ivarA > ivarB then ;if A is greater than B play 440Hz
    al oscil 10000,440,1
  elseif ivarA < ivarB then ;if A is less than B play 220Hz
    al oscil 10000,220,1
  endif
  out al
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1
i1 0 10
</CsScore>
</CsoundSynthesizer>
Chapter 2
First steps, orchestra, score

2.1 Amplitude, frequency
2.2 0dbfs
2.3 Fundamental waveforms
2.4 Disk rendering
2.5 P-fields
2.6 Methods for events management
2.1 Amplitude, frequency

To generate white noise (plotted in the previous example), the orchestra used a `rand` opcode controlled by a single **Amplitude** parameter.

```plaintext
asignal rand 10000 ; (10000 = amplitude value)
```

As can be easily guessed in Csound many other opcodes exist, and their number of parameters varies according to the complexity of the object itself. We will now consider one of the most important Csound opcode, the opcode `oscil`. Take a look at the syntax in detail:

```plaintext
asig oscil iamp,ifreq,ifn
```

How to read the syntax? If we take a look at the official Csound user handbook, we know that we can describe the opcode `oscil` as a sound module to generate a waveform (but we will see later that it holds many uses, for example the construction of control signals); its parameters are, in order: amplitude, frequency and function number.

A possible example of an orchestra:

```plaintext
instr 1 ; instrument name
asound oscil 10000,220,1 ; asound variable
out asound ; asound out bus (mono)
endin ; end line
```

We have just created an instrument that can generate a waveform with amplitude 10000 and frequency of 220 Hz. Which waveform is this? Is the opcode `oscil` related to a particular type of waveform? From the informations we gather from our “instr 1” we are not able yet to understand if the synthesis process will determine a square wave, a sawtooth wave, a triangular one or something else.

At this point we introduce one of the main features of this programming language: **functions** (used in Csound not only to generate waveforms). Observing the third parameter of our opcode:

```plaintext
asound oscil 10000,220,1
```

the last parameter, with the value of “1” is related to a function, in this case this value refers to a particular type of waveform that can be found in the score:

```plaintext
f1 0 4096 10 1 ; f1 generates a simple sine wave
```

a full example:

```plaintext
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
```
In this first introductory section we used `oscil` to generate a waveform, this choice has mainly historical reasons. Classic examples of this language as the immense Csound catalog (an instruments catalog related to the text Csound book) use this historical Csound opcode included since the earliest versions. There are many implementations of this opcode but the ”standard” choice is the high-precision `poscil`. In the section of Canonical Manual named Basic Oscillators different types can be found:

- simple oscillator (like `oscil`)
- linear interpolation oscillator (the name ends with letter “i”)
- cubic interpolation oscillator (the name ends with number “3”)
- oscillator bank (`oscbnk`)

`poscil` is similar to `oscil` but offers more precise control over frequency, working with long wavetables (functions) and very low frequency values with extreme precision, examples in this book will be based entirely on this opcode. `poscil` also accepts negative values for the frequency, this aspect makes it very suitable for AM and FM synthesis (discussed in later chapters) and allows to work with tables that are not based on the “exponentiation of 2”.

### 2.2 0dbfs

One of the simplest methods to manage the amplitude level of a Csound file is the 0dbfs opcode, corresponding to the greater amplitude value in digital audio before clipping.

0dbfs = 32767 (bipolar range of values for a 16 bit or a 16 bit AD/DA codec audio file)
This opcode can be declared in the header:

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

in this way amplitude values in a range between 0 e 1 can be assigned.
In later chapters this opcode will be used.

**Notes**

If, for example, we set `0dbfs = 20` we obtain a range between 0.0 and 20, and so on, outside of this range (0 to 20) we naturally get a distortion effect. Be careful to control the amplitude of your instruments to save your speakers (in addition to your precious ears)

---

Example Chap 2.1

```xml
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

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

instr 1
asound poscil .5,220,1
out asound
endin

</CsInstruments>
<CsScore>

i1 0 4
f1 0 4096 10 1
</CsScore>
</CsoundSynthesizer>
```

Let’s try to synthesize the sound for listening to it, as in the previous example we save our files, in WinXound choose Tools from the menu and click on Compile (or click on the green button similar to a play button), if all goes well we will be able to listen to four seconds of a sinusoid with a frequency of 220 hz.

We have thus introduced a major new element of Csound called function (f1 0 4096 10 1), let’s observe observe in detail the structure of our function:
Example Chap. 3.22_Generative3

```xml
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1 ;control instrument for instr 2, instr 3
indx = 1 ;(indx = indx + incr, indx < inumber trigger the loop)
incr = 1
inumber = 200 ;number of iterations
instrument1 = 2 ;instrument number (p1)
instrument2 = 3 ;instrument number (p1)
iamp = .6 ;oscillator amplitude
ifreq1 = 1000 ;frequency1
ifreq2 = 20 ;frequency2
idelay = 0 ;attack (p2)
idur = .1 ;event duration (p3)

sequence: ;starts the loop
ifreq1 = ifreq1 - 5 ;generates a descending scale (increase = 5)
ifreq2 = ifreq2 + 5 ;generates an ascendant scale (increase = 5)
idelay = idelay + .1 ;time interval between events

event_i "i",instrument1,idelay,idur,iamp,ifreq1;scales with different attack

loop_lt indx,incr,inumber,sequence ;generates sequence

instr 2 ;generator controlled by instr 1
a1 poscil p4/2,p5,1
kenv linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0
out a1*kenv

instr 3 ;generator controlled by instr 1
a1 pluck p4/2,p5,p5*2,0,1
kenv linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0
out a1*kenv

</CsInstruments>
<CsScore>
```

89
Now another example with loop_lt, the result is a random melody (based on a scale with a defined pitch) in continuous time variation

Example Chap.3.23_neverending_melody

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
  sr  =  44100
  kr  =  4410
  ksmps = 10
  nchnls = 2
  0dbfs = 1
  gifn ftgen 1,0,1024,10,1
  gifn ftgen 2,0,1024,10,1,1
  gifn ftgen 3,0,1024,10,1,0,1
  gifn ftgen 4,0,1024,10,1,0,0,1
  gifn ftgen 5,0,1024,10,1,0,0,0,1

  instr 1
    indx = 1
    incr = 1
    iattack = 0
    loop:
      idur = rnd(1)
      iattack = iattack+idur
      event_i "i",2,iattack,idur
      loop lt indx,incr,1000,loop
    endin

  instr 2
    iscale = 200 ;pitches table
    kmelody = rnd(15) ;choose a note from the table
    kpitch table kmelody+1,iscale ;reads the note function
    kamp = rnd(1) ;generates random amplitude values
    ifn = rnd(4) ;choose a different waveform for each event
    a1 poscil .1+kamp,cpspch(kpitch),ifn+1 ;oscillator
    aenv line .7,p3,0 ;amplitude envelope
    krndpan = rnd(1) ;random values for pan
    a1,ar pan2 a1,krndpan ;random panning
    outs a1*aenv,ar*aenv ;enveloped stereo output
  endin
```
Finally we use the loop technique to generate a high number of sine waves, it’s a simple example of additive synthesis in which melodies are randomly generated at different speeds.

Example Chap. 3.24_200oscil

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

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

seed 0 ;global random, generates a new random value every time a file is played  

instr 1  
instrument = 2  
inum=1  
loop:  
iamp random .1,.6 ;amplitude values  
ifreq random 100,12000 ;frequency values  
inumber = p4 ;number of oscillators  
irate random .1,20 ;melody speed  
event_i "i",instrument, 0, p3,iamp/(p4*.3),ifreq,irate  
loop_lt inum,1,inumber,loop  
endin  

instr 2 ;additive synthesis generator  
kfreq randomh 60, p5, p6 ;random melody al poscil p4, kfreq,1 ;generates an oscillator bank  
kpan randomi 0,1,4  
al,ar pan2 al,kpan  
kenv linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0 ;declick envelope  
```

Chapter. 3 *Main techniques*
There are different ways to define a function in Csound as well as the classic one inside the score. For example, you can define a table directly in the orchestra as in the following example:

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
  sr = 44100
  kr = 4410
  kbpsms = 10
  nchnls = 1
  0dbfs = 1
  gisine ftgen 1,0,16384,10,1

  instr 1
    al poscil 1,220,gisine
    out al
  endin
</CsInstruments>
<CsScore>
  i1 0 5 1
</CsScore>
</CsoundSynthesizer>
```

Or by defining the table inside an instrument using the opcode `ftgenonce`, the latter case can greatly help in terms of clarity of the code. Consider, however, that a function inserted into an instrument is not “global”, but is read only by the instrument that contains it.
A possible use:

```xml
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

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

instr 1

gisine ftgenonce 1,0, 16384,10,1
a1 poscil 1,220,gisine
out a1
endin

</CsInstruments>
<CsScore>

i1 0 5 1

</CsScore>
</CsoundSynthesizer>

The table gisine that uses the opcode `ftgenonce` can be read only by the instrument.
Chapter 4
Csound: MIDI and Osc

4.1 Introduction to MIDI-opcodes
4.2 Virtual MIDI keyboard
4.3 A first MIDI synth
4.4 MIDI controller
4.5 Read MIDI files
4.6 Microtonal scales
4.7 Complex events generation
4.8 Csound and OSC
4.1 Introduction to MIDI-opcodes

One of the main reasons that often kept aspiring electronic musicians away from Csound has always been interactivity. For several reasons Csound isn’t a language suitable for real-time purposes. The first problem was the computing power that did not allow the compilation and listening in real time, especially with very complex orchestras and synthesis. The historical 1992 Csound version (Barry Vercoe) introduced for the first time the MIDI opcodes to enable the communication with the outside world via the MIDI protocol, but unfortunately, at that time, the only machines capable of carrying out such a huge calculation amount were some expensive workstations.

With the gradual increase of the computing power of personal computers and with the addition of several new opcodes and flags to manage the real-time, over the last decade has become an interactive instrument. One of the keys was the release of Direct Csound by Gabriel Maldonado (June 1998). It was an enhanced version for live performance and synchronized with the Csound version 3.494. In January 2002, the project was named CsoundAV whose last release was in 2005 (synced to the Csound version 4.23 with graphical interfaces support, OpenGL for graphics and numerous other innovations). The power of these versions is the chance to interface directly with your audio converter (Motu, Rme, Apogee, Focusrite, etc..) Using, for example, (in the case of Windows machines) Asio drivers to reduce latency during live performances.

The recent standard Csound versions (from 5 onwards) integrate almost all the opcodes of CsoundAV (except for OpenGL) and it’s to this version we refer to in book examples - for portability and to take advantage of future features of this language, since the wonderful CsoundAV project is no longer evolving.

What does concretely mean using Csound in real time? The only limit is the imagination of course, we can control the sound synthesis parameters by moving the mouse with the virtual sliders, or by using an external MIDI controller. We can enable or disable specific functions, process an audio file in real time, saving on the hard disk the audio file of our live performances in any format even multichannel, we can process a signal picked up by a microphone and then make the perfect integration between acoustic instruments and electronic world.

The list of Csound MIDI opcodes is very large and includes, among many others, the following characteristics:

- send pitch, velocity, aftertouch, pitch bend, MIDI channel number, program change, etc..
- define alternative tuning systems (like microtones)
- receive external MIDI controller data of any kind
- write Csound orchestras with interoperability between MIDI and score
- send messages to
4.2 Virtual MIDI keyboard

One of the most practical tools for managing the virtual controller MIDI data is introduced in Csound version 5.07 Steven Yi. It’s a virtual MIDI keyboard with total MIDI channels control, program change and slider fully configurable, the virtual MIDI keyboard is available in both Csound5 distributions for PC and Mac. Run the following program from winxound:

Example Chap. 4.1_virtual MIDI keyboard

```c
<CsoundSynthesizer>
<CsOptions>
 -+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>
 instr 1
 ;instrument design....
 endin
 </CsInstruments>
<CsScore>
 i1 0 3600
 </CsScore>
</CsoundSynthesizer>
```

Our virtual controller appears:
Let’s now analyze the short script:

```xml
<CsOptions>
  -+rtmidi=virtual -M0
</CsOptions>
```

The Csound flags have to be written between these tags (CsOptions). The flag `rtmidi` activates the virtual MIDI keyboard, the flag “-M” activates the MIDI input port (in this case number 0 refers to the virtual keyboard, but if you have other controllers: such as the MIDI ports of your sound card or other connected USB MIDI controller, you will need to set a different number, for example-M1-M2, etc.).

```xml
to display the virtual keyboard Csound requires the writing of at least one code block with its `endid` tool (in our case it’s an empty instrument), although our instrument will not produce any kind of sound we still have to activate it from the score for any period of time (p3) in order to activate the virtual MIDI controller.
```

```xml
<i1 0 3600 ;instrument 1 active for 1 hour.
</CsScore>
```

### 4.3 A first MIDI synth

The first practical example to control Csound via MIDI will be the modification of one of the previous chapters examples, we will build a synthesis instrument whose frequency will be controlled in real time by MIDI events (in this case, the keys of the virtual keyboard), to do this we will use a special opcode `cpsmidi`.

```xml
Example Chap. 4.2_oscmidi_A

```xml
<CsoundSynthesizer>
<CsOptions>
  -odac -+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
Odbfs = 1
instr 1
  iamp ampmidi .6
  ifreq cpsmidi
  ifunc = rnd(8)
al poscil iamp,ifreq,ifunc+1
```
Please note: we used as envelope a variant of linseg that suits MIDI performance, linsegr.
The code line “a1 oscili iamp, ifreq, ifunc +1” is controlled in amplitude and frequency by two new opcodes ampmidi (which contains velocity values) and cpsmidi (MIDI note number converted in hertz) that return amplitude and frequency. The third parameter of poscil “ifunc” is controlled by a random number generator in order to generate a waveform always different in each MIDI event. Finally we applied a simple envelope “declick.” to the audio output.
It can be easily noticed that the approach from the musical composition point of view has changed dramatically, our score is no longer defined according to fixed parameters, it becomes interactive. In this way it’s possible to “play” Csound as a standard commercial hardware or software synthesizer. Our score in this case will check only the duration of the MIDI performance.
Now let’s see a variant of the previous example in which you can extend, over the release, the duration of a played note. The variable kbend defines the range of the pitchband function of a MIDI master keyboard (pitch = ifreq + kbend).
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kbend pchbend    -1,1

iamp ampmidi .3

ifreq    cpsmidi

ifunc    =    rnd(8)

al    poscil iamp,ifreq+kbend,ifunc+1

irel    =    2    ;note duration

kdeclick    linsegr 1,1,0,irel,0

out    al*kdeclick

endin

</CsInstruments>
</CsScore>

An alternative technique to control MIDI is MIDI of inter-op command line flags. These flags replace ampmidi and cpsmidi opcodes, allowing you to control in real time an instrument created for the control via score. Therefore, the command line will become:

<CsoundSynthesizer>
<CsOptions>
   -rtmidi=virtual -midi-key-cps=5 -midi-velocity-amp=4 -M0
</CsOptions>
</CsInstrumenets>

line that represents the following control statements:

- **cps=5**: indicates parameter p5 (same function as cpsmidi)
- **amp=4**: indicates parameter p4 (same function as ampmidi)
Example Chap. 4.2_oscilMIDI_C

<CsoundSynthesizer>
<CsOptions>
  -rtmidi=virtual  -midi-key-cps=5  -midi-velocity-amp=4  -M0
</CsOptions>
<CsInstruments>
  sr = 44100
  kr = 4410
  ksmpls = 10
  nchnls = 1
  0dbfs = 1
  instr 1
    ifunc = rnd(8)
    a1 poscil p4,p5,ifunc+1
    kenv madsr .2,.8,.5,.2
    out a1*kenv
  endin
</CsInstruments>
<CsScore>
  f1 0 4096 10 1
  f2 0 4096 10 0 1
  f3 0 4096 10 0 1 1
  f4 0 4096 10 1 0 0 1
  f5 0 4096 10 0 1 0 1
  f6 0 4096 10 0 1 0 0 1
  f7 0 4096 10 1 0 1 0 0 1
  f8 0 4096 10 1 0 1 0 1 0 1 0 1
  f9 0 4096 10 0 1 0 1 0 1 0 1 0 1
  f0 3600
</CsScore>
</CsoundSynthesizer>

You can also specify a MIDI channel number for a specific instrument, for example it may be useful to change the MIDI channel from your master keyboard to play a different instrument. To do this we use `massign`:

```
massign ichnl, insnum
massign ichnl, “insname”
```

The second opcode allows you to use a “name” for the instrument to be activated, before instruments declaration in the orchestra `massign` must be entered as a constant, ie:
4.4 MIDI controller

In the next example we introduce the use of particular opcode suitable to send data via a MIDI controller number and MIDI channel preset. It is ctrl7 (of which there are extended versions as ctrl14 and ctrl21), in our example this opcode will be associated with a virtual MIDI slider, here is the ctrl7 syntax:

```
kdest ctrl7 ichan, ictlno, kmin, kmax
```

ichan: MIDI channel number (1-16)
ictlno: MIDI controller number (0-127)

kmin, kmax: values range

ctrl7 can be our virtual controller or any external hardware controller. To use one of the many MIDI controllers on the market it’s sufficient to indicate the flag-M “number” the MIDI in port number of another controller. How do we know the exact number with which our machine displays various MIDI ports? We try to give a very high number to the flag-M, for example,-M88, thereby Csound will return an error because certainly, on our computer, we do not have 88 MIDI controllers! However, you can display in the console window precise informations about MIDI devices installed on your machine (this system can also be used for the flag-ODAC that manages audio devices) and then we will select the correct external MIDI controller number.
Example Chap. 4.3_ctrl7

```csound
<CsoundSynthesizer>
<CsOptions>
  -rtmidi=virtual -M0
</CsOptions>

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

; midi channel set to 1 for each controller
instr 1
  kamp  ctrl7 1,1,0,1 ; range between 0 and 0.7 maximum instrument value
  kfreq1 ctrl7 1,2,20,2000 ; controller 2 with range 20
  kfreq2 ctrl7 1,3,20,2000 ; controller 3 with range 20
  kfreq3 ctrl7 1,4,20,2000 ;
  kfreq4 ctrl7 1,5,20,2000 ;
  kfreq5 ctrl7 1,6,20,2000 ;
  kfreq6 ctrl7 1,7,20,2000
  kfreq7 ctrl7 1,8,20,2000
  kfreq8 ctrl7 1,9,20,2000
  kfreq9 ctrl7 1,10,20,2000

  a1 poscil kamp,kfreq1,1 ; kamp e kfreq managed by MIDI controller
  a2 poscil kamp,kfreq2,1
  a3 poscil kamp,kfreq3,1
  a4 poscil kamp,kfreq4,1
  a5 poscil kamp,kfreq5,1
  a6 poscil kamp,kfreq6,1
  a7 poscil kamp,kfreq7,1
  a8 poscil kamp,kfreq8,1
  a9 poscil kamp,kfreq9,1

  asum sum a1,a2,a3,a4,a5,a6,a7,a8,a9 ; sums audio variable
  aout asum/9
endin
</CsInstruments>

<CsScore>
f1 0 4096 10 1
i1 0 3600 ; instrument 1 turned on for one hour
</CsScore>
```

105
In this example of additive synthesis, we created an instrument with simple sine waves. The variable kamp controls the overall amplitude of the instrument (slider 1), the frequencies will be controlled by the slider of the virtual keyboard (slider controller number from 2-10).

As you can see the creative possibilities are endless. We try to build an instrument with amplitude, frequency, vibrato and auto-pan speed controlled in real time by MIDI virtual sliders.

Example Chap. 4.4_vibratoMIDI

```<CsoundSynthesizer>
<CsOptions>
-odac  -+rtmidi=virtual  -M0
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1
instr 1
;canale midi = 1
;slider 1 = vco oscillator amplitude
;slider 2 = vco oscillator frequency
;slider 3 = vibrato amplitude
;slider 4 = vibrato frequncy
;slider 5 = autopan rate
```
Chapter 4 Csound: MIDI and Osc

```
kamp ctrl7 1,1,0,.6 ;amplitude range 0 1000
kfreq ctrl7 1,2,20,2000 ;oscillator frequency
kampvib ctrl7 1,3,1,20 ;vibrato amplitude
kfrequvib ctrl7 1,4,.1,20 ;vibrato frequency
kvib poscil kampvib,kfrequvib,1 ;vibrato
al poscil kamp,kfreq+kvib,1 ;audio variables
krate ctrl7 1,5,.1,3 ;controls autopan speed
kautopan poscil .5,krate,1 ;autopan
aleft = al * (1-kautopan) ;Pan Left
aright = al * kautopan ;Pan Right
outs aleft, aright
```

```CsInstruments>
</CsInstruments>
<CsScore>
f1 0 14096 10 1 0 .4 0 .3
i1 0 3600 ;instrument one turned on for one hour
</CsScore>
</CsoundSynthesizer>

4.5 Read MIDI files

MIDI opcodes as well as receiving MIDI data from external controllers can even import and read MIDI files. Csound provides the-F flag to read a MIDI file:

```
<CsOptions>
-odac -M0 -F “fileName”
</CsOptions>
```

With this command, we are enabling an audio port (ODAC) and a MIDI input (-M), the line of code following the name of the file that Csound will play in real time. With recent versions you can also manage multi-channel MIDI files. In this example, the MIDI file will be in the same folder as the .Csd, but we can also specify any directory.

With what timbre our MIDI files will play? This will of course depend on chosen synthesis method. The next example will use an emulation of plucked string (pluck). We will also see an example of hard disk recording with which Csound write a stereo audio file of our MIDI performance.
We analyze some important fragments of this example, we have created an instrument “1” (plucked string) with amplitude and frequency controlled via MIDI. The second instrument acts as hard disk recorder and runs a rendering on disk of the MIDI performance (in the case of reading a MIDI file). We see the use of variables outside the instrument block: they are global variables (must start with “g”) accessible from all instruments of the orchestra. Instr 1 uses the opcode `vincr` to sum two
signals. What are the two signals in this example? This is the variable a1 (pluck) added to the global variable.

```
vincr gaout1,a1
vincr gaout2,a1
```

now the signal recorded in `vincr` goes to instr 2:

```
fout $FILENAME,4,gaout1,gaout2
```

Opcode `fout` writes an audio file on our hard drives and the number 4 indicates the selected file type, `fout` offers plenty of choice. Let’s deepen the *Canonical Reference*:

0 - 32-bit floating point samples without header (binary PCM multichannel file)

1 - 16-bit integers without header (binary PCM multichannel file)

2 - 16-bit integers with a header. The header type depends on the render (-o) format. For example, if the user chooses the AIFF format (using the -A flag), the header format will be AIFF type.

3 - u-law samples with a header (see iformat=2).

4 - 16-bit integers with a header (see iformat=2).

5 - 32-bit integers with a header (see iformat=2).

6 - 32-bit floats with a header (see iformat=2).

7 - 8-bit unsigned integers with a header (see iformat=2).

8 - 24-bit integers with a header (see iformat=2).

9 - 64-bit floats with a header (see iformat=2).

Finally we find the opcode `clear` that serves to avoid cumulative effects rescaling global variables to the value of zero (corresponding to the statement `gaout = 0`).

### 4.6 Microtonal scales

We introduce an important opcode with which you can define alternative tuning systems. In the previous examples we used the opcode `cpsmidi` (MIDI note number converted in hertz) to control the synthesis methods through a MIDI keyboard. Suppose we want to divide the octave of our master keyboard using, for example, quarter tones, so smaller intervals than tones and semitones belonging to Western culture. With Csound you can write unusual tuning systems using the opcode `cpstmid`, its syntax:

```
icps cpstmid ifn
```

`ifn` is linked to a table where we define the interval relations of our micro-tonal scale. How `cpsmidi` also `cpstmid` was created to receive data from MIDI instruments. To define the micro-tonal scale is necessary to indicate some initial parameters such as the number of notes in the scale, its base frequency, etc., In order:

- `numgrades`: number of scale notes;
• *interval*: scale range, for example 2 to indicate an octave;
• *basefreq*: base frequency of the scale in Hz;
• *basekeyMIDI*: MIDI note associated to the base frequency;

After the four fundamental parameters, you can define the individual interval ratios. Here is an example of a simple table for a traditional scale of 12 equal semitones:

| f1   | 0   | 64  | -2 | 12 | 2  | 60 | 1  | 1.059463094359 | 1.122462048309 | 1.189207115003..etc... |

We see a complete example of real time synthesis with the use of microtonal scales, the tables that define the systems of alternative scales have been for convenience written in the orchestra using the *ftgen* opcode (that allows functions definition inside the ‘orchestra’).

In order we can find the following scales: *Standard tuning, Quarter tones, Decatonic, Pythagorean, Third tones, Detuned, Harmonic, Chinese, Major Triad, Fibonacci, Pentatonic, Bohlen-Pierce, Bharata tuning, Indian shruti, Wendy Carlos Alpha.*

---

```<CsoundSynthesizer>
<CsOptions>
  -rtmidi=virtual  -M0
</CsOptions>

<s=44100
.kr = 4410
.ksmps = 10
.nchnls = 2
.0dbfs = 1

#include "tuning.h"

;101 Standard tuning
;102 Quarter tones
;103 Decatonic
;104 Pythagorean
;105 Third tones
;106 Detuned
;107 Harmonic
;108 Chinese
;109 Major Triad
;110 Fibonacci
;111 Pentatonic
;112 Bohlen-Pierce
;113 Bharata tuning
;114 Indian shruti
;115 Wendy Carlos Alpha
```
Chapter 4 Csound: MIDI and Osc

%instrument 1
instr 1
 iamp ampmidi .6
 ifreq cpstmid 112 ;table number referring to the micro-
 tonal scale
 a1 pluck iamp,ifreq,220,0,1
 kenv linsegr 0,.1,1,.1,.1,.0
 outs a1*kenv,a1*kenv
 endin
</CsInstruments>
<CsScore>
f1 0 4096 10 .9 .8 .7 .6 .5 .4 .3 .2 .1
f0 3600
</CsScore>
</CsoundSynthesizer>

4.7 Complex events generation

In the Generative Processes section of the third chapter we introduced some tech-
niques based on an instrument that controls another one. Imagine to apply this con-
cept to a MIDI performance, for example to build an FX processor with MIDI delay
lines, or generate a list of MIDI data triggered by the master keyboard.

A first approach may be using the opcode event_i: let’s build a control structure
composed of several instances of this opcode. Each instance will have the same at-
tack value but with a smooth pitch transition for each event. The result will then be
given by the sum of several instances of the same note but with variations in pitch,
similarly to the result produced by a signal processor commonly called chorus.

Example Chap. 4.7_MIDI_chorus

<CsoundSynthesizer>
<CsOptions>
 -rtmidi=virtual -M0 ;activates the midi virtual keyboard
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1
massign 1,1 ;assigns midi channel 1 to instr 1
instr 1
 giamp ampmidi .2
idur = 2 ; events duration
gifreq = cpsmidie
idelay = 0 ; attack
instrument = 2 ; instrument to be controlled

; several “instr 2″ instances with transposition

event_i “i”, instrument, idelay, idur, giamp, gifreq
event_i “i”, instrument, idelay, idur, giamp, gifreq+2
event_i “i”, instrument, idelay, idur, giamp, gifreq+4
event_i “i”, instrument, idelay, idur, giamp, gifreq+6
event_i “i”, instrument, idelay, idur, giamp, gifreq+8
event_i “i”, instrument, idelay, idur, giamp, gifreq+10
event_i “i”, instrument, idelay, idur, giamp, gifreq-2
event_i “i”, instrument, idelay, idur, giamp, gifreq-4
event_i “i”, instrument, idelay, idur, giamp, gifreq-6
event_i “i”, instrument, idelay, idur, giamp, gifreq-8
event_i “i”, instrument, idelay, idur, giamp, gifreq-10
endin

instr 2 ; plucked string

a1 pluck p4, p5, i(gifreq), 0, 1
irel = 2
kenv mxadsr 0.01, .8, .4, 1
outs a1*kenv, a1*kenv
endin

</CsInstruments>
<CsScore>

f0 36000
</CsScore>
</CsoundSynthesizer>

In this instrument we have the variable:

gifreq = cpsmidie

whose value increases or decreases each control instance for the instrument “2” (gifreq +2, gifreq +3, +4 gifreq, etc.). The sum of the various events (with the same attack value) creates a sort of “jarring” effect, precisely due to pitch variations indicated as constant in the instrument 1.

With this simple approach we can experience a great number of live performance. Let’s try for example to indicate different attack values for each control instance, or imagine to control different instruments.

The instrument that follows introduces a variation of the previous instrument. This time, the component of pitch transpose is randomly generated for each trigger of the instrument. The same procedure will be used to control random parameter “attack”.

In this way we build a sort of MIDI delay with a different result at each event.
Example Chap. 4.8_MIDI_delay

```xml
<CsoundSynthesizer>
<CsOptions>
  -rtmidi=virtual -M0
</CsOptions>

<CsInstruments>
  sr = 44100
  kr = 4410
  ksmps = 10
  nchnls = 2
  0dbfs = 1
  massign 1,1
  gifn ftgen 1,0,1024,10,1

instr 1
  giamp ampmidi .2
  gifreq cpsmidi ;base pitch
  idur = 2 ;events duration
  idevpitch1 = gifreq+birnd(10)
  idevpitch2 = gifreq+birnd(10)
  idevpitch3 = gifreq+birnd(10)
  idevpitch4 = gifreq+birnd(10)
  idevpitch5 = gifreq+birnd(10)
  idevpitch6 = gifreq+birnd(10)
  idevpitch7 = gifreq+birnd(10)

  idelay1 = 0
  idelay2 = rnd(1)
  idelay3 = rnd(1)
  idelay4 = rnd(1)
  idelay5 = rnd(1)
  idelay6 = rnd(1)
  idelay7 = rnd(1)
  idelay8 = rnd(1)

  instrument = 2 ;controls instrument 2

;various instances of “instr 2” with transpositions and different attack values
  event_i "i",instrument,idelay1,idur,giamp,gifreq
  event_i "i",instrument,idelay2,idur,giamp,idevpitch1
  event_i "i",instrument,idelay3,idur,giamp,idevpitch2
  event_i "i",instrument,idelay4,idur,giamp,idevpitch3
  event_i "i",instrument,idelay5,idur,giamp,idevpitch4

</CsInstruments>
</CsoundSynthesizer>
```
We introduce one last example with some parameters controllable in real time. Using the loop technique in Csound, with a few lines of code, we can create a large number of events (the only limit is obviously the computing power of our machine). The following example generates a complex rhythmic sequence with three control parameters for real time associated with three sliders:

- slider 1: pitch transpose of each event
- slider 2: delay time (delay)
- slider 3: random delay amount (to generate micro rallentandi or accelerandi).

This time we will write a single code line using the opcode `event_i`

```plaintext
event_i "i", instrument, idelay, idur, giamp, gifreq, inumber, iselect
```

The parameter `inumber` will control the instance number of this statement with a loop:

```plaintext
loop: ; the section to reiterate begins

gifreq = gifreq*iratio ; pitch controlled by slider 1
idelay = idelay+itime+rnd(irndtime) ; delay controlled by slider 2 and 3
iselect = 1 ; oscillator selection from instr 2
```

```plaintext
endin
```

```plaintext
endin
```

```plaintext
</CsInstruments>
</CsScore>
</CsoundSynthesizer>
```

```plaintext
f0 36000
</CsScore>
</CsoundSynthesizer>
```

We introduce one last example with some parameters controllable in real time. Using the loop technique in Csound, with a few lines of code, we can create a large number of events (the only limit is obviously the computing power of our machine). The following example generates a complex rhythmic sequence with three control parameters for real time associated with three sliders:

- slider 1: pitch transpose of each event
- slider 2: delay time (delay)
- slider 3: random delay amount (to generate micro rallentandi or accelerandi).

This time we will write a single code line using the opcode `event_i`

```plaintext
event_i "i", instrument, idelay, idur, giamp, gifreq, inumber, iselect
```

The parameter `inumber` will control the instance number of this statement with a loop:

```plaintext
loop: ; the section to reiterate begins

gifreq = gifreq*iratio ; pitch controlled by slider 1
idelay = idelay+itime+rnd(irndtime) ; delay controlled by slider 2 and 3
iselect = 1 ; oscillator selection from instr 2
```

```plaintext
endin
```

```plaintext
endin
```

```plaintext
</CsInstruments>
</CsScore>
</CsoundSynthesizer>
```

```plaintext
f0 36000
</CsScore>
</CsoundSynthesizer>
```
Example Chap. 4.9_Hiperpluck

<CsoundSynthesizer>
<CsOptions>
-rtmidi=virtual -M0
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1
instr 1

giamp ampmidi 1 ;midi control
gifreq cpsmidi

instrument = 2 ;controlled instrument
inum=1 ;to activate the loop section
idelay = 0
inumber = 20 ;number of events to be generated
idur = 2 ;event duration
iratio ctrl7 1,1,1,1.2 ;pitch transpose
itime ctrl7 1,2,.001,.1 ;delay between an event and its next
irndtime ctrl7 1,3,.001,.1 ;randomly controls delay time

loop: ;trigger the loop section

gifreq = gifreq*iratio ;pitch controlled by slider 1
idelay = idelay+itime+rnd(irndtime) ;delay controlled by slider 2 and 3
iselect = 1 ;oscillator selection from instr 2

event_i “i”,instrument,idelay,idur,giamp,gifreq,inumber,iselect

loop_le inum,1,inumber,loop ;repeats the process for 20 instances(inumber)
endin

instr 2

kselect = p7 ;oscillator selection (pluck, poscil)

if kselect = 1 then
    a1 pluck p4/p6, p5, 1000, 0, 1
elseif kselect = 2 then
    a1 poscil p4/p6, p5, 1
endif

kenv adsr 0.001,.4,.5,.1 ;envelope
outs a1*kenv,a1*kenv
endin
4.8 Csound and OSC

OSC protocol (Open sound control) is a communication system between electronic instruments, much more recent of MIDI system, was developed at CNMAT by Adrian Freed and Matt Wright. It’s a transmission protocol in which data passes through a network (TCP/IP, Internet, etc.). The advantage over the MIDI system concerns, among other things, the largest number of data exchange (1024 messages) and lower latency. However, we don’t necessarily think of OSC in reference (or as a substitute) to the MIDI protocol, since this is still a valid and stable system to exchange data between electronic devices. A typical performance with electronic instruments that communicate via OSC uses wireless controller, and new generation control systems (eg, tablet pc) without using cables for the connection between instruments. This opens up vast possibilities of research in the multimedia filed. Imagine the control of a synthesis engine by a motion sensor applied on the body of an actor/dancer.

In the latest release of Csound were introduced various types of opcodes for the control via OSC. The applications in the hardware world ranging from the use of tablet pcs up to games controllers like the Nintendo Wii remote controller. The following example will use the following opcodes:

```
ihandle OSCinit iport
kans OSClisten ihandle, idest, itype[, xdata1, xdata2, …]
```

Let’s see a typical Csound architecture that uses OSC protocol:

```
;port number for the communication between Csound and an external device
giOsc1 OSCinit 5000
instr 1
kslider init 0 ;starting value
k1 OSClisten giOsc1, ”/1/fader1”, ”i”, kslider ;control slider
a1 oscili iamp, kslider, 1
```

...
The next example will be based on communication system between Csound and a mobile control device such as Iphone, Ipad. With such devices (as well as alternative devices based on Android operating system) is possible to build a custom OSC controller, using software tools like Touch Osc (http://hexler.net/software/touchosc). For first you install on your computer Touch Osc Editor and create a new project:

![Touch Osc Editor](image)

The display panel is a graphical environment for drawing custom control interfaces, Touch Osc contains several widgets such as sliders, buttons, joysticks, etc.. We choose a simple slider clicking the right mouse button (select Fader V or Fader H). In the panel on the right we can write the identifying name of the graphic object, which will be associated with the Csound control instrument. In this case the object Fader will take the default name:

/l/fader1

Csound orchestra will communicate with the control interface as follows:

k1 OSClisten giOsc1,"/l/fader1","i",kslider

In the range field we put the minimum and the maximum values.
An example: let’s open the file foscili.touchosc from Touch Osc editor to activate the sending process to the mobile device via the Sync function. Let’s then open Touch Osc from your mobile device. In the main panel there’s the field Connections, open the “OSC” item and write the IP number of the computer on which Csound is running. Let’s modify the following settings in the opened window:

- host : computer IP (example 192.168.1.3)
- enabled OK
- port Outgoing 5000 (number of the communication port)
- port incoming 0 (default)
Chapter 4  
Csound: MIDI and Osc

Let’s close the window and go back to the main panel to the Layout item. Let’s select the first item in the list, Add. Touch Osc will make a small scan of connected devices (Found Host ()), selecting the name of your computer (remember the editor in Sync mode), you can import the file to foscili.touchosc inside our device.

At this point let’s activate our custom controller imported from the Layout list, if all goes well you should see on the mobile device the controller previously designed with the editor.

Let’s analyze the following Csound file, Cap.4.7_fosciliOSC and let’s try to send data via Touch Osc, the synthesis type chosen as example is the frequency modulation technique, which will be discussed in later chapters.

Example Chap. 4.7_fosciliOSC

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
    sr = 44100
    kr = 4410
    ksmps = 10
    nchnls = 1
    0dbfs = 1

    giOsc1 OSCinit 5000 ; Touch OSC communication port
   instr
   kcar init 0 ; variables initialisation
   kmod init 0
   kindx init 0
   kamp init 0

   k1 OSClisten giOsc1, "/1/fader1", "I", kcar
   k2 OSClisten giOsc1, "/1/fader2", "I", kmod
   k3 OSClisten giOsc1, "/1/fader3", "I", kindx

   a1 foscili .6,1,kcar,kmod,kindx,1
   out a1

endin
</CsInstruments>
<CsScore>
    f1 0  4096 10   1
    i1 0  3600 ; instrument active for one hour
</CsScore>
</CsoundSynthesizer>
```
We made a simple communication system based on OSC. Softwares as TouchOSC allow the creation of control instruments, peculiar and customizable overcoming the obvious limitations of commercial hardware controllers, born with specific characteristics and so limited.
Chapter 5
Graphical User Interfaces
(Fltk, Max/Msp)

5.1 The Fltk Library
5.2 Fltk panel
5.3 Fltabs
5.4 Knobs, sliders, buttons, counter, Joysticks
5.5 Automations, MIDI control
5.6 Snapshots
5.7 Graphics library with QuteCsound
5.8 Csound~ and Max/Msp
5.9 CsoundForLive
5.10 Cabbage and Vst development
5.1 The Fltk Library

In this section we will introduce the techniques for building GUIs (Graphical User Interfaces) to make our interactive virtual instruments. Csound5 integrates a specific library of graphic objects called Fltk (Fast Light Tool Kit), this is a library fast, powerful, multi-platform with a relatively simple syntax. With this library, we can build graphic tools to control synthesizers, samplers and signal processors.

Fltk is a Csound integrated library that contains a high number of opcodes, and allows the creation of a single .csd file that integrates graphics and synthesis engine. This choice is certainly a good solution from the code portability point of view, however, it must be clear that this type of choice, over the years, has not satisfied a part of Csound programmers. Recent trends (not to be considered as standard choices) and programming modes, prefer the separation of Csound code from graphics management, such as with the use of specific tools like CsoundQt, Cabbage or languages such as Max / MSP, Java, etc. Therefore, the discussion in a comprehensive way of graphic components in Csound is a very complex matter in continuous development that makes it difficult, if not impossible, for a full discussion.

Fast Light Tool Kit is mainly divided in **Containers** and **Valuators**:

<table>
<thead>
<tr>
<th>Containers:</th>
<th>Valuators:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Panels</td>
<td>• Sliders</td>
</tr>
<tr>
<td>• Scroll areas</td>
<td>• Knobs</td>
</tr>
<tr>
<td>• Pack</td>
<td>• Rollers</td>
</tr>
<tr>
<td>• Tabs</td>
<td>• Text fields</td>
</tr>
<tr>
<td>• Groups</td>
<td>• Joysticks</td>
</tr>
<tr>
<td></td>
<td>• Counters</td>
</tr>
</tbody>
</table>

Let’s see the skeleton of a csd file to understand the basic rules of graphic objects: each Fltk opcode has to be inserted after the initial header. You can not use Fltk inside an instrument.

```<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

Flpanel "Text field",iwidth,iheight,ix,iy
FlpanelEnd
Flrun ;initializes the graphic thread
instr 1
```

122
What we see is the basic structure to build a window inside a Csound instrument. **Fpanel** **FpanelEnd** delimit the *container*. You can initialize several containers then define the size and the coordinates in the space. Defined containers shall start with the graphic engine **Flrun**. As we have seen in the section relating to the MIDI, also in this case the task of the score is to hold the instance of our instrument “on” (f0 3600).

### 5.2 Fltk panel

Let’s see a practical **Flpanel** example to draw two windows of the same size and with different colors. The Flpanel syntax:

```
Flpanel “testo”,iwidth,iheight [, ix][, iy][, iborder][, ikbd-capture][, iclose]
```

The basic parameters are the first four (sizes and coordinates), the other parameters are optional:

- **IBorder**: it’s possible to choose the border thickness (from 0 to 7).
- **Ikbd-capture**: capture keyboard events, with value = 0 disables the function iClose: set to value = 1 prevents the window from closing using the close button, a very important function especially when working with many windows.

To modify the window color we use the opcode:

```
Flcolor ired,igreen,iblue
```

Flcolor sets primary colors in RGB values (it’s recommended to consult a table to set different RGB colors).

---

Example Chap. 5.1 _Flpanel_

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
iwidth  = 500 ;width
iheight = 500 ;height
ix     = 50 ;x and y coordinates for space positioning
```
iy = 50
iborder = 5 ;black line border
ikbdcapture = 0 ;disable events capture from keyboard
iclose = 1 ;disable the window close button
;first window
Flcolor 0,250,0 ;RGB window color
Flpanel "FLTK panel 1",iwidth,iheight,ix,iy,iborder,ikbdcapture,iclose
FlpanelEnd
;
;second window
Flcolor 110,180,210
Flpanel "FLTK panel 2",iwidth,iheight,ix+520,iy,iborder,ikbdcapture,iclose
FlpanelEnd
Flrun ;initialize the graphic thread
instr 1
; ..........
endin
</CsInstruments>
<CsScore>
f0 3600 ;real time performance duration
</CsScore>
</CsoundSynthesizer>
Chapter. 5 Graphical User Interfaces (Fltk, Max/Msp)

5.3 Fltabs

We now introduce the Fltabs opcode with which you can organize multiple windows within a single container (Flpanel). The various overlapping windows will be accessed via buttons, the structure to display two overlapping and accessible windows is represented by the following scheme:

```
Flpanel "Multipanel", 500, 500, 100, 100
Fltabs itabswidth, itabsheight, ix, iy
; first window
FGroup "Panel1", itabwidth, itabheight, ix, iy
FGroup_end
; second window
FGroup "Panel2", itabwidth, itabheight, ix, iy
FGroup_end
FltabsEnd
FGroupEnd
Frun
```

**Fltabs** is a useful interface to view different areas that alternate in the same window (initially created by Flpanel), the opcode **FGroup** describe the area within the main window. We observe that the entire structure is opened and closed by the code block between Flpanel and FlpanelEnd, now a complete example:

```
Example Chap. 5.2_Multiwindows

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

Flpanel "Multiwindows", 500, 500, 100, 100
itabswidth = 500
itabsheight = 500
ix = 5
iy = 5
Fltabs itabswidth, itabsheight, ix, iy
; first window
itab1width = 500
itab1height = 500
itab1x = 10
itab1y = 40
FGroup "Panel1", itab1width, itab1height, itab1x, itab1y
FGroup_end
```

125
Chapter 12 Signal processors

12.3 Chorus

In this part of the DSP chapter, we will examine some of the most important signal processings.

The effect called “chorus” is a type of processing in which the signal is added to several instances of the same signal with slight variations caused by variable delay lines and with tonal variations. To create a chorus effect there is a specific opcode ability to produce a variable delay in time, \texttt{vdelay}.
ares vdelay asig, adel, imaxdel

• asig: input signal
• adel: delay in ms
• imaxdel: max delay value in ms

A simple example of stereo chorus: there are eight variable delays (divided between left and right) with controls on depth and speed (similar to chorus pedals used by electric guitarists):

Example Chap. 12.6_chorus

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
chnlns = 2
0dbfs = 1

instr 1
a1 diskin2 "guitar.wav", 1, 0, 1
idepth = p4 ; dry/wet
irate = p5 ; delay speed
k1 randomi .01, idepth, irate
adel1 vdelay a1, k1, 1000
k2 randomi .01, idepth*0.1, irate*0.1
adel2 vdelay a1, k2, 1000
k3 randomi .01, idepth*0.2, irate*0.2
adel3 vdelay a1, k3, 1000
k4 randomi .01, idepth*0.3, irate*0.3
adel4 vdelay a1, k4, 1000
k5 randomi .01, idepth*0.4, irate*0.4
adel5 vdelay a1, k5, 1000
k6 randomi .01, idepth*0.5, irate*0.5
adel6 vdelay a1, k6, 1000
k7 randomi .01, idepth*0.6, irate*0.6
adel7 vdelay a1, k7, 1000
k8 randomi .01, idepth*0.7, irate*0.7
adel8 vdelay a1, k8, 1000
aenv linseg 0, 0.02, 1, p3 - 0.05, 1, 0.02, 0, 0.01, 0

aenv linseg 0, 0.02, 1, p3 - 0.05, 1, 0.02, 0, 0.01, 0
Chapter 12 Signal processors

outs ((adel1+adel2+adel3+adel4)*aenv)*.3,((adel5+adel6+adel7 +adel8)*aenv)*.3
endin
</CsInstruments>
<CsScore>
i1 0 4 .1 .3
s
i1 0 4 1 2
s
i1 0 4 4 4
s
i1 0 4 5 16
s
i1 0 4 18 18
s
</CsScore>
</CsoundSynthesizer>

12.4 Flanger

The **flanger** technique has its origin in the analog world. It consisted in play two tapes of the same recorded material. While playing, one of the two tapes was pressed in some points generating variations in the reading speed. The other tape was mixed to the tape subjected to this treatment, obtaining this particular effect.

Csound has his own effect called flanger:

```
ares flanger asig,adel,kfeedback
```

- **asig**: input signal
- **adel**: delay in seconds
- **kfeedback**: feedback value (range between 0 and 1)

An example of stereo **flanger** in which two delay lines are controlled by a low frequency oscillator. The control parameters relate to the density, the LFO speed rate and the amount of feedback.

```
Example Chap. 12.7_flanger
```

```xml
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
```
0dbfs = 1

instr 1
a1 vco2 .2,55,0 ;sawtooth
a2 vco2 .2,220,0
asum sum a1,a2
idepth = p4 ;depth
irate = p5 ;LFO speed rate
adel1 poscil idepth/2000,irate/2000,1
adel2 poscil idepth/2000,irate/1000,1
kfeedback = p6 ;feedback
aflang1 flanger asum,adel1,kfeedback
aflang2 flanger asum,adel2,kfeedback
outs (aflang1+asum)*.5,(aflang2+asum)*.5 ;original signal + effect
endin
</CsInstruments>

Another possible variant with four flanger instances, each one with random values for the delay parameter.

Example Chap. 12.8_flanger_quad

 sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1
Chapter 13
U.D.O opcodes

13.1 Hipersaw U.D.O.
13.2 U.D.O. Chorus
13.3 Chowning, Risset Libraries
13.4 Vocal synthesis library
13.1 Hipersaw U.D.O.

UDO opcode technique allows to customize your own Csound modules. These opcodes are directly written in Csound language (not to be confused with standard opcodes in C language with which Csound is built). The unbelievable advantage of this technique is to enclose, encapsulate, reuse in a simple and immediate way, entire pieces of code. For example we may want to turn an instrument with a particular synthesis technique into an opcode, a piece of code capable of performing a complex mathematical calculation, a signal processor, etc..

Imagine a simple orchestra as the following one, we have a waveform inspired by the famous Roland’s supersaw, substantially a 7 sawtooth waves bank, each one with his own pitch detune

Example Chap. 13.1_Hypersaw

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
sr = 44100
;kr = 4410
ksmps = 16
nchnls = 1
0dbfs = 1

instr 1

iamp = p4
kdetune = p6

krnd1 poscil kdetune,.1,1
krnd2 poscil kdetune,.2,1
krnd3 poscil kdetune,.3,1
krnd4 poscil kdetune,.4,1
krnd5 poscil kdetune,.5,1

a1 vco2 iamp,p5,0
a2 vco2 iamp,p5+kdetune,0
a3 vco2 iamp,p5+(kdetect+krnd1)*.1,0
a4 vco2 iamp,p5+(kdetect+krnd2)*.1,0
a5 vco2 iamp,p5+(kdetect+krnd3)*.1,0
a6 vco2 iamp,p5+(kdetect+krnd4)*.1,0
a7 vco2 iamp,p5+(kdetect+krnd5)*.1,0

asum sum a1,a2,a3,a4,a5,a6,a7

kcut = p7
kreso = p8
</CsInstruments>
Appendix

Csound opcodes list

The opcodes list (updated to 6.02 version). To get the list there is a specific flag called 

\texttt{-z}.

\begin{tabular}{lll}
ATSadd & ATSaddnz & ATSbufread \\
ATScross & ATSinfo & ATSinterpred \\
ATSpartialtap & ATSread & ATSreadnz \\
ATSSinnoi & FLbox & FLbutBank \\
FLTbutton & FLcloseButton & FLcolor \\
FLcolor2 & FLcount & FLEXecButton \\
FLgetsnap & FLgroup & FLgroupEnd \\
FLgroup \_end & FLhide & FLhvsBox \\
FLhvsBox\_setValue & FLjoy & FLkeyIn \\
FLknob & FLLabel & Fllloadsnap \\
FLmouse & FLpack & FLpackEnd \\
FLpack \_end & FLpanel & FLpanelEnd \\
FLpanel \_end & FLprintk & FLprintk2 \\
FLRoller & FLrun & FLsavesnap \\
FLscroll & FLscrollEnd & FLscroll\_end \\
FLsetAlign & FLsetBox & FLsetColor \\
FLsetColor2 & FLsetFont & FLsetPosition \\
FLsetSize & FLsetSnapGroup & FLsetText \\
FLsetTextColor & FLsetTextSize & FLsetTextType \\
FLsetValue & FLsetVal & FLsetVal_i \\
FLsetsnap & FLshow & FLslidBnk \\
FLslidBnk2 & FLslidBnk\_set & FLslidBnk\_setk \\
FLslidBnk\_getHandle & FLslidBnk\_set & FLslidBnk\_setk \\
FLslider & FLtabs & FLtabs\_end \\
FLtabs \_end & FLtext & FLupdate \\
FLvalue & FLvkeybd & FLvslidBnk \\
FLvslidBnk2 & FLxyin & MixerClear \\
MixerGetLevel & MixerReceive & MixerSend \\
MixerSetLevel & MixerSetLevel\_i & OSCinit \\
MixerSetLevel\_i & & OSCsend \\
OSC\_listen & OSCrecv & STKBlowBotl \\
STKBandedWG & STK\_BeeThree & STKBrass \\
STK\_BlowHole & STK\_Bowd & STKFVoices \\
STK\_Clarinet & STK\_Drummer & STKMandolin \\
STK\_Flute & STK\_HeavyMetal & STK\_PercFlut \\
STKModalBar & STK\_Moog & STKRhodey \\
STK\_Plucked & STK\_Resonate & STKS\_Simple \\
STK\_Saxophone & STK\_Shakers & STK\_TubeBell \\
STK\_Sitar & STK\_StifKarp & STK\_Wurley \\
STK\_VoicForm & & active \\
\texttt{abs} & & adsyn \\
\texttt{add} & & \\
\end{tabular}
Thanks

First of all, I have to thank the amazing Csound community, developers, musicians, sound designers such as John Ffitch, Victor Lazzarini, Andres Cabrera, Gabriel Maldonado (CsoundAV), Steven Yi (with its amazing Blue), Michael Gogins, Iain McCurdy Rory Walsh (and his amazing Cabbage), Stefano Bonetti (author of the famous Winxound editor) and many, many others. Without their immense research work this book would never seen the light!

A special thanks to Dr. Richard Boulanger for encouragement, enthusiasm, and passion given in almost a decade.

Finally, I thank the people who in one way or another supported me during this long process of writing, in particular Emanuela Arancio, Paola Bozzalla, Giancarlo Zedde, Amedeo Casella, Salvatore Livecchi, Alessandro Merlo, Antonello Mattana, Simone Offredo, Alessandro Rossini, Carlo Panebianco, Alessandro Biglione, Pierpaolo Barbano, Alessandro Di Paola, Davide Ficco, Gianluca Verlingieri, Giuseppe Elos, Mauro Bouvet, Stefano Bassanese, Valentino Zorino, Giuseppe Bordonaro, Eduard Arancio, Carlo Matti, Claudio Ribezzo.

Special thanks to my parents.