Scripting Csound 5

Victor Lazzarini
Music Technology Laboratory
National University of Ireland, Maynooth
Victor.Lazzarini@nuim.ie

Abstract

This article introduces a scripting environment for Csound 5. It introduces the Csound 5 API and discusses it use in the development of a TclTk scripting interface, TclCsound. The three components of TclCsound are presented and discussed. A number of applications, from simple transport control of Csound to client-server networking are explained in some detail. The article concludes with a brief overview of some other Csound 5 language APIs, such as Java and Python.

Keywords: Music Programming Languages; Scripting Languages; Computer Music Composition.

1 Introduction

The Csound music programming system (Vercoe 2004) is currently the most complete of the text-based audio processing systems in terms of its unit generator collection. Csound hails from a long tradition in Computer Music. Together with cmusic (Moore 1990), it was one of the first modern C-language-based portable sound compilers (Pope 1993), when it was released in 1986. Due to its source-code availability, first from the cecelia MIT ftp server and then from the DREAM site at Bath, it was adopted by composers and developers world-wide. These brave new people helped it to develop into a formidable tool for sound synthesis, processing and computer music composition. Its latest version, Csound 5 (ffitch, 2005) has close to one thousand opcodes, ranging from the basic table lookup oscillator to spectral signal demixing unit generators.

Many important changes have been introduced in Csound5, which involved a complete redesign of the software. This resulted not only in a better software, from an engineering perspective, but in the support for many new possible ways of using and interacting with Csound.

An important development has been the availability of a complete C API (the so-called 'Host API', which was, in fact, already partially present in earlier versions). The API can be used to instantiate and control Csound from a calling process, opening a whole new set of possibilities for the system.

2 The Csound 5 API

The Csound 5 Host API allows the embedding of the audio processing system under other 'host' software. Effectively, Csound is now a library, libcsound, that can provide audio services, such as synthesis and processing, for any application. This allows for complete control over the functioning of the audio engine, including transport control, loading of plugins, inter-application sofware bus, multithreading, etc.. A 'classic' csound command-line program can now be written based only on a few API calls:

```
#include <csound.h>
int main(int argc, char **argv) {

int result;
CSOUND *cs; /* the csound instance */

/* initialise the library */
csoundInitialize(&argc, &argv, 0);
/* create the csound instance */
cs = csoundCreate(NULL);
/* compile csound code  */
result = csoundCompile(cs, argc, argv);
/* this is the processing loop */
if(result) while(csoundPerformKsmps(cs)==0);
/* destroy the instance */
csoundDestroy(cs);
return 0;
}
```

The Csound API can be used in many applications; the development of frontends is the most obvious of these. A good example of its application is found on the csoundapi~ Class, which provides a multi-instantiable interface to Csound 5 for Pure Data. The Csound API is the basis for TclCsound (Lazzarini 2005), a Tcl/Tk extension, discussed in the next section.

3 TclCsound

The classic interface to csound gives you access to the program via a command-line such as

```
csound -odac hommage.csd
```

This is a simple yet effective way of making sound. However, it does not give you neither flexibility nor interacion. With the advent of the API, a lot more is possible. At this stage, TclCsound was introduced to provide a simple scripting interface to Csound. Tcl is a simple language that is easy to extend and provide nice facilities such as easy file access and TCP networking. With its Tk component, it can also handle a graphic and event interface. TclCsound provides three 'points of contact' with Tcl:

1. a csound-aware tcl interpreter (cstclsh)
2. a csound-aware windowing shell (cswish)
3. a csound commands module for Tcl/Tk (tclcsound dynamic lib)

3.1 The Tcl interpreter: cstclsh

With cstclsh, it is possible to have interactive control over a csound performance. The command starts an interactive shell, which holds an instance of Csound. A number of commands can then be used to control it. For instance, the following command can compile csound code and load it in memory ready for performance:

```
csCompile -odac hommage.csd -m0
```

Once this is done, performance can be started in two ways: using `csPlay` or `csPerform`. The command

```
csPlay
```

will start the Csound performance in a separate thread and return to the cstclsh prompt. A number of commands can then be used to control Csound. For instance,

```
csPause
```

will pause performance; and

```
csRewind
```

will rewind to the beginning of the note-list. The csNote, csTable and csEvent commands can be used to add Csound score events to the performance, on-the-fly. The `csPerform` command, as opposed to `csPlay`, will not launch a separate thread, but will run Csound in the same thread, returning only when the performance is finished. A variety of other commands exist, providing full control of Csound.

3.2 Cswish: the windowing shell

With Cswish, Tk widgets and commands can be used to provide graphical interface and event handling. As with cstclsh, running the cswish command also opens an interactive shell. For instance, the following commands can be used to create a transport control panel for Csound:

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

Similarly, it is possible to bind keys to commands so that the computer keyboard can be used to play Csound.

Particularly useful are the control channel commands that TclCsound provides. For instance, named IO channels can be registered with TclCsound and these can be used with the invalue, outvalue opcodes. In addition, the Csound API also provides a complete software bus for audio, control and string channels. It is possible in TclCsound to access control and string bus channels (the audio bus is not implemented, as Tcl is not able to handle such data). With these TclCsound commands, Tk widgets can be easily connected to synthesis parameters.

3.3 A Csound server

In Tcl, setting up TCP network connections is very simple. With a few lines of code a csound server can be built. This can accept connections from the local machine or from remote clients. Not only Tcl/Tk clients can send commands to it, but TCP connections can be made from other sofware, such as, for instance, Pure Data (PD). A Tcl script that can be run under the standard tclsh interpreter is shown below. It uses the Tclcsound module, a dynamic library that adds the Csound API commands to Tcl.

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

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

# this arranges for connections to be made
```

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

# this sets up a server to listen for
# connections
set server [socket -server NewChan 40001]
set sinfo  [fconfigure $server -sockname]
puts "Csound server: ready for connections on
port [lindex $sinfo 2]"
vwait forever
```

With the server running, it is then possible to set up clients to control the Csound server. Such clients can be run from standard Tcl/Tk interpreters, as they do not evaluate the Csound commands themselves. Here is an example of client connections to a Csound server, using Tcl:

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

# compile Csound code
puts $sock "csCompile -odac hommage.csd"
flush $sock

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

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

As mentioned before, it is possible to set up clients using other software systems, such as PD. Such clients need only to connect to the server (using a netsend object) and send messages to it. The first item of each message is taken to be a command. Further items can optionally be added to it as arguments to that command.

3.4 A Scripting Environment

With TclCsound, it is possible to transform the popular text editor e-macs into a Csound scripting/performing environment. When in Tcl mode, the editor allows for Tcl expressions to be evaluated by selection and use of a simple escape sequence (ctrl-C ctrl-X). This facility allows the integrated editing and performance of Csound and Tcl/Tk code.

In Tcl it is possible to write score and orchestra files that can be saved, compiled and run by the same script, under the e-macs environment. The following example shows a Tcl script that builds a csound instrument and then proceeds to run a csound performance. It creates 10 slightly detuned parallel oscillators, generating sounds similar to those found in Risset's *Inharmonique*.

```
load tclcsound.so Tclcsound

# set up some intermediary files
set orcfile "tcl.orc"
set scofile "tcl.sco"
set orc [open $orcfile w]
set sco [open $scofile w]

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

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


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

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

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

# build the instrument and a dummy score
MakeIns 1 $ins
puts $sco  "f0 10"

close $orc
close $sco

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

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

# send in a sequence of events and perform it
for {set i 0} { $i < 60 } { incr i } {

    csNote 1 [expr $i * 0.1] .5 \
      [expr ($i * 10) + 500] [expr 100 + $i *
10]

}

csPerform

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

The use of such facilities as provided by e-macs can emulate an environment not unlike the one found under the so-called 'modern synthesis systems', such as SuperCollider (SC). In fact, it is possible to run Csound in a client-server set-up, which is one of the features of SC3. A major advantage is that Csound provides about three or four times the number of unit generators found in that language (as well as providing a lower-level

approach to signal processing, in fact these are but a few advantages of Csound).

3.5 TclCsound as a language wrapper

It is possible to use TclCsound at a slightly lower level, as many of the C API functions have been wrapped as Tcl commands. For instance it is possible to create a 'classic' Csound command-line frontend completely written in Tcl. The following script demonstrates this:

```
#!/usr/local/bin/cstclsh
set result 1
csCompileList $argv
while { $result != 0 } {
set result csPerformKsmps
}
```

This script is effectively equivalent to the C program shown in section 2. If saved to, say, a file called csound.tcl, and made executable, it is possible to run it as in

```
csound.tcl –odac hommage.csd
```

4 OTHER LANGUAGE WRAPPERS

It is very likely that many users will prefer to run Csound from their programming environment of choice. For these, C++, Lisp (using CFFI), Java and Python are other available languages. Access to the Csound library is provided by SWIG-generated wrappers.

4.1 Python and Java examples

The way these languages interact with the Csound 5 library is very similar to the C API. A major difference is that they are require to import the Csound module (based on a 'native' library module), called `csnd`. In Python, the csnd.py and csnd.pyc files, distributed with Csound, hold the Python API, whereas in Java, the csnd.jar archive holds the csnd package.

Generally, for a straight performance, the steps involved are:

1. Create a Csound instance:

Java:
```
cs = new Csound();
```

Python:
```
cs = csnd.csoundCreate(None);
```

2. Compile Csound code:

Java:
```
cs.Compile("hommage.csd");
```

Python:
```
csnd.csoundCompile(cs, 2,
       ['csound', 'hommage.csd'])
```

3. Run a processing loop:

Java:
```
int result;
while(result == 0)
   result = cs.PerformKsmps();
```

Python:
```
while result == 0:
   result =csoundPerformKsmps(cs);
```

4. Clean up, ready for a new performance:

Java:
```
cs.Reset();
```

Python:
```
csnd.csoundReset(cs);
```

The Java and Python wrappers open up many new possibilities for using Csound programmatically. There are, though, a few aspects of the C API, which are very C-specific and do not translate into Java or Python. However, these should not make any impact on the majority of the applications that the system might have.

5. Conclusion and Future Prospects

Csound is a very comprehensive synthesis and musical signal processing environment. The additional facilities provided in its latest version have brought it up-to-date with more modern software engineering concepts. Its existence as a library has enabled a variety of new uses and added new 'entry-points' into the system. It is very likely that such enhancements will also spur further features, esp. when the new parser for the language, under development by John ffitch, is introduced. It will then be possible to develop other ways of interacting with the system, such as alternative synthesis languages, interpreters and further facilities for networking and distributed operation.

5 References

John ffitch. On the design of Csound 5. *Proceedings of the 2005 Linux Audio Conference*,ZKM, Karlsruhe, Germany.

Victor Lazzarini. The TclCsound frontend and language Wrapper. *Sounds Electric 2005*. NUI, Maynooth, Ireland.

F Richard Moore. 1990. *Elements of Computer Music*, Englewood Cliffs, NJ: Prentice-Hall, 1990.

Stephen T Pope. 1993. Machine Tongues XV: Three Packages for Software Sound Synthesis. *Computer Music Journal 17 (2).*

Barry Vercoe et Al. 2005. *The Csound Reference Manual*.
http://www.csounds.com/manual/html/index.html