Guide to DVD Chapter 35 Examples: *Rory Walsh*

Developing Audio Software with the Csound Host API

This 'how-to' guide takes you through the steps involved in building the source code that accompanies DVD Chapter 35, *Developing Audio Software with the Csound Host API*. The examples have been tested on OS X, Linux and Windows. All the examples were built using the GNU C and C++ compilers. Please read all the instructions carefully before attempting to build the sample applications.

In order to build any of the examples provided you must have Csound installed on your machine. The latest version of Csound for OSX provides users with everything they need to start developing Csound API applications. Whenever you build a Csound host application you will need to pass the header file paths to your compiler. The header files will be in folder called "Headers". In order for the compiler to be able to locate these header files we use the *-I* flag like this:

```
-I/Library/Frameworks/CsoundLib.framework/Versions/Current/Headers
```

We also need to tell the compiler where to find the Csound libraries, and this is typically specified using the following command-line flag:

```
-L/Library/Frameworks/CsoundLib.framework/Versions/Current/
```

This also applies to all the paths found in the two Scons build scripts that come with these examples. The paths contained within those files will also have to be updated to reflect where you have installed Csound.

The GNU compilers should be installed by default on your OS X machine, if they are not they can be downloaded from http://developer.apple.com

Setting up the Csound API Libraries

When you install Csound using one of the OS X installers (http://csound.sourceforge.net) it automatically installs the main Csound framework *csoundLib.* Contained within that framework are all the libraries you will need to build Csound host applications.

Building the Command-line Applications

There are two ways to build the command-line applications found in this folder. The first is to build them one by one from the command-line and the second is to use Scons. More details on using Scons are provided below.

Building the Command-line Applications: Method 1

Remember to make sure that you are in the correct directory before trying to build any of the example applications. Below are the commands you will use on OS X to build the command line examples manually. Pay particular attention to the fact that when compiling .cpp files we must use the g++ compiler and when compiling .c files we use the gcc compiler. Remember to provide the correct paths to the Csound 'H' and 'interfaces' folders as described in the opening paragraphs. Also ensure that you specify the correct path to the folder containing the Csound import libraries.

To compile examples 1 through 10 you will use the following command-lines.

```
gcc example1.c -o example1 -I/Library/Frameworks/CsoundLib.framework/Versions/Current/Headers
-framework CsoundLib  -L/Library/Frameworks/CsoundLib.framework/Versions/Current/ -l_csnd

gcc example2.c -o example2 -I/Library/Frameworks/CsoundLib.framework/Versions/Current/Headers
-framework CsoundLib  -L/Library/Frameworks/CsoundLib.framework/Versions/Current/ -l_csnd

gcc example3.c -o example3 -I/Library/Frameworks/CsoundLib.framework/Versions/Current/Headers
-framework CsoundLib  -L/Library/Frameworks/CsoundLib.framework/Versions/Current/ -l_csnd

etc...
```

Building the Command-line Applications: Method 2

The second method for building the command line applications uses Scons, an open source cross platform software construction tool. Scons uses the Python programming language therefore both will have to be installed before use. Python can be downloaded and installed from www.python.org while Scons can be downloaded from www.scons.org. As Scons is written in Python, a prior knowledge of Python could prove useful, but it is not essential. Scons scripts are provided for both the command line applications and the wxWidgets GUI applications.

Here's a fully commented version of the Sconstruct file used for building the command line applications. Remember that the scripts provided with the examples won't work unless you update the include and library paths to point towards the *Header* folder and the folder that contains your Csound import libraries.

```
# SConstruct for Developing Audio Applications with the Csound Host API
# examples and projects
# (c) R Walsh, 2009

import os
import sys

# Simple function to Detect OPERATING SYSTEM.
def getPlatform():
    if sys.platform[:5] == 'linux':
        return 'linux'
    elif sys.platform[:3] == 'win':
        return 'win32'
    elif sys.platform[:6] == 'darwin':
        return 'darwin'
    else:
        return 'unsupported'

print "\nPlatform is: "
print getPlatform()
print "\n"

#add paths for csound header files, these need to be edited according to
where you have installed Csound!
if getPlatform() == 'win32':
      #set the environment for the build and tell scons to use mingw. Scons
      #will use MSVC by default if both MSVC and mingw are installed
      env = Environment(tools = ['mingw'], ENV = {'PATH' :
os.environ['PATH'], 'TEMP' : os.environ ['TEMP']})
      #alter this path so it points to where you have the Csound  and
wxWidgets libs and headers
      env.Append(LIBPATH = ['C:/MyDocuments/SourceCode/Csound5/csound5'])
      env.Append(CPPPATH =
['C:/MyDocuments/SourceCode/Csound5/csound5/H','C:/MyDocuments/SourceCode/Cso
und5/csound5/interfaces'])
elif getPlatform() == 'linux':
      env =  Environment(ENV = {'PATH' : os.environ['PATH']})
      env.Append(LIBPATH = ['/home/rory/SourceCode/csound/csound5'])
      env.Append(CPPPATH = ['/home/rory/SourceCode/csound/csound5/H',
'/home/rory/SourceCode/csound/csound5/interfaces'])
elif getPlatform() == 'darwin':
      env =  Environment(ENV = {'PATH' : os.environ['PATH']})
      env.Append(CPPPATH =
['/Library/Frameworks/CsoundLib.framework/Versions/Current/Headers'])
      env.Append(LIBPATH =
['/Library/Frameworks/CsoundLib.framework/Versions/Current/'])

#add linker options here, they will differ for each different OS
if getPlatform() == 'linux':
      # csound lib
      env.Append(LIBS = ['csound', 'pthread', 'dl', 'sndfile', 'm', 'csnd'])
elif getPlatform() == 'win32':
      # csound libs
      env.Append(LIBS = ['libcsound32','libcsnd'])
elif getPlatform() == 'darwin':
      # csound framework
      env.Append(LINKFLAGS = ['-framework', 'CsoundLib'])
      env.Append(LIBS = ['_csnd'])

#check to see that our header files are present and can be found by the
compiler
configure = env.Configure()
sane = configure.CheckHeader("stdio.h", language="C")
if not sane:
      print "\n*** BUILD ERROR: there is a problem with your C/C++ compiler"
      print "  =>please check it before proceeding\n"
#     sys.exit()

csound = configure.CheckHeader("csound.h", language="C")
if not csound:
      print "\n***BUILD ERROR:csound header not found"
      print "\n   Make sure you have specified the correct path"
      print "\n   to your Csound5 'H' directory and the Csound"
      print "\n   interfaces directory"
#     sys.exit()

if sane and csound:
 print "\n*** BUILDING PROGRAMS:"
 print "     example1, example2, example3, example4, example5"
 print "     example6, example7, example8, example9, example10"

example1 = env.Program('example1', 'example1.c')
example1 = env.Program('example2', 'example2.c')
example1 = env.Program('example3', 'example3.c')
example1 = env.Program('example4', 'example4.c')
example1 = env.Program('example5', 'example5.c')
example1 = env.Program('example6', 'example6.c')
example1 = env.Program('example7', 'example7.cpp')
example1 = env.Program('example8', 'example8.cpp')
example1 = env.Program('example9', 'example9.cpp')
example1 = env.Program('example10', 'example10.cpp')

# you can add here your own projects here
# use this format:
# project = env.Program('project', 'source.c')
# where project is the name of your new program
# and source
```

You are encouraged to simply add your own projects to this file in the future once your paths are set up correctly. This will save you the trouble of having to create a new Scons file for every project. Scons, when evoked from the command line, will always read a file called SConstruct by default. As we don't have any scripts/files called SConstruct we will need to tell Scons what files to read. We can do this by using the `-f` flag to specify a particular file. For example:

```
scons -f SConstructCommandLine
```

This command will use the above script to build all the command-line examples contained in this folder. Once again, be careful to insure that you have specified the correct paths in your Sconstruct file and that you are calling Scons from the correct folder, i.e., the one containing the examples.

**Note that sometimes old object files might cause problems when compiling new source code, for this reason it's a good idea to periodically clean your install directory by passing a '--clean' to your Scons command.*

Building the GUI Examples

In order to build the GUI applications you will need to download and install wxWidgets. (www.wxWidgets.org) You will need to build the libraries yourself. To do this simply unzip the compressed file to a directory. Open your shell/terminal and do the following:

```
$ cd into the wxWidgets base dir
$ mkdir osx-build
$ cd osx-build
$ ../configure --disable-shared
$ make
$ sudo make install
```

The recommended way to build the GUI examples is with Scons. This is because of the number of libraries you need to link against. To build, run Scons with the SConstructGUI script:

```
scons -f SconstructGUI
```

As you can see from looking at the command given below, Scons is by far the easiest way to build the GUI applications. If however you wish to build each one manually you can pass the following command line arguments to you MSYS shell. Don't forget to update the paths to reflect your system:

```
g++ helloworld.cpp -o HelloWorld -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -
D__WXDEBUG__ -D__WXMAC__ -
I/Library/Frameworks/CsoundLib.framework/Versions/Current/headers -
I/usr/include -I/usr/lib/wx/include/mac-unicode-debug-2.8 -I/usr/include/wx-
2.8 -framework CsoundLib -
L/Library/Frameworks/CsoundLib.framework/Versions/Current -l_csnd -lwx_macud-
2.8 -framework QuickTime -framework IOKit -framework Carbon -framework Cocoa
-framework System System

g++ GUIexample1.cpp -o GUIexample1 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -
D__WXDEBUG__ -D__WXMAC__ -
I/Library/Frameworks/CsoundLib.framework/Versions/Current/headers -
I/usr/include -I/usr/lib/wx/include/mac-unicode-debug-2.8 -I/usr/include/wx-
2.8 -framework CsoundLib -
L/Library/Frameworks/CsoundLib.framework/Versions/Current -l_csnd -lwx_macud-
2.8 -framework QuickTime -framework IOKit -framework Carbon -framework Cocoa
-framework System System

g++ GranulatorV1.cpp -o GranulatorV1 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -
D__WXDEBUG__ -D__WXMAC__ -
I/Library/Frameworks/CsoundLib.framework/Versions/Current/headers -
I/usr/include -I/usr/lib/wx/include/mac-unicode-debug-2.8 -I/usr/include/wx-
2.8 -framework CsoundLib -
L/Library/Frameworks/CsoundLib.framework/Versions/Current -l_csnd -lwx_macud-
2.8 -framework QuickTime -framework IOKit -framework Carbon -framework Cocoa
-framework System System

g++ GranulatorV2.cpp -o GranulatorV2 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -
D__WXDEBUG__ -D__WXMAC__ -
I/Library/Frameworks/CsoundLib.framework/Versions/Current/headers -
I/usr/include -I/usr/lib/wx/include/mac-unicode-debug-2.8 -I/usr/include/wx-
2.8 -framework CsoundLib -
L/Library/Frameworks/CsoundLib.framework/Versions/Current -l_csnd -lwx_macud-
2.8 -framework QuickTime -framework IOKit -framework Carbon -framework Cocoa
-framework System System
```

Once you have built the GUI examples there is only one step remaining in order to get them to run properly and that's to add them to a *bundle*. A *bundle* is nothing more than a directory that contains all the files needed to run an OS X application. A full exploration of OS X Bundles is beyond the scope of this text, however, bundles have been provided for each of the GUI examples. All we have to do is to copy the applications to the appropriate bundles. The easiest way to do this is from the command line using the following commands:

```
cp HelloWorld ./HelloWorld.app
cp GUIexample1 ./GUIexample1.app
cp GranulatorV1 ./GranulatorV1.app
cp Granulatorv2 ./GranulatorV2.app
```

Once the binaries have been placed in their respective bundles you can launch the applications by double clicking their icon from within the examples folder. For those of you with some prior knowledge of python you can automate this process described above by adding some system functions to your Scons script so that the files get placed into their respective folders during building.

Overview of the Example Applications

example1: This example recreates the classic Csound command-line interface. You must pass a valid *.csd* file to this application on startup. For example:

```
$./example1 myfile.csd
```

example2: This example illustrates how to communicate on a named channel between a host application and an instance of Csound using the ***chnget*** opcodes.

```
$./example2
```

example3: This example illustrates how to communicate on a named channel between a host application and an instance of Csound using the ***invalue*** opcodes.

example4: This example illustrates how to set up a custom Csound thread.

example5: This example shows how to send score events to an instance of Csound. When prompted users should enter a score event with a start time, a duration, an amplitude value(0-32000) and a pitch value in Hz. For example:

```
$  0 5 10000 440
```

example6: In this example, we see how one can use low-level C functions to generate a score and write it to disk for Csound to read.

example7: This example demonstrates the Csound C++ class.

example8: This example illustrates the use of the `CsoundPerformanceThread` class.

example9: This example shows how to use `Csound::SetChannel()` to communicate with an instance of Csound. Press any number between 1 and 5 (followed by Enter) to start a pattern of notes. To stop the program simply hit Ctrl+C or enter a number higher than 5.

example10: This example shows how to use the `CppSound` class and some of it's useful member functions. Press 1 (followed by Enter) to start playback; Press 2 (followed by Enter) to stop and Ctrl+C to exit.

HelloWorld: This example does not create any audio. It simply demonstrates a most basic wxWidgets application with some simple menu commands.

GUIexample1: A GUI version of example9. Go to the **File -> Start** menu to start the audio.

GranulatorV1: This is a realtime granular synthesis instrument. Make sure you have a microphone set up; otherwise you can pass a sound file to the soundin opcode in *granny.csd.*

Go to the **File -> Start** menu to begin processing audio.

GranulatorV2: Similar to the GranulatorV1 but with more features and more efficient coding. Make sure you have a microphone set up, otherwise, as above pass a sound file to the soundin opcode in *granulatorV2.csd* (in the bundle). Go to **File -> Start** to begin processing audio. This example also illustrates how to read a .csd file that is embedded in an OS X bundle. Note, if you wish to change the .csd so that it supports sample processing vs. real-time audio processing, you will need to edit the *granulatorV2.csd* file in the bundle. (Ctrl+Click on *GranulatorV2.app* and select Show Package Contents from the menu).