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 versions of Csound for windows provide users with everything they need to start developing Csound API applications. Whenever you build a Csound host application you will need to pass the paths to the header files to your compiler. The header files will be in folders called *H* and *interfaces* that reside in the Csound installation directory. By default this will be `C:/Program Files/Csound`. To provide the compiler with the paths to the correct folders we use the *-I* flag like this:

`-I"C:/Program Files/Csound/H"` and
`-I"C:/Program Files/Csound/interfaces"`

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.

**Please ensure that the CsOptions for each of the Csound files in these examples are correct for your system. This means making sure you use the correct buffer sizes and audio devices. If they are not set properly your applications will either quit at run-time or have poor quality audio. Also take care if copying and pasting commands from this document to the command line as the certain punctuation marks in the `Courier New` typeface pose problems for some compilers.*

To build these example you can use MinGW/MSYS(www.mingw.org) or Cygwin. www.cygwin.com. Note that although Cygwin provides a more comprehensive environment most users will find that MinGW/MSYS provides them with more than enough to compile and build most open source applications.

Setting up the Csound API Libraries

When you install Csound using one of the windows installers (http://csound.sourceforge.net) it

automatically installs the main Csound library *csound32.dll.5.2*[1] and the auxiliary interfaces library *csnd.dll* in the Csound *bin* folder. You have two choices when it comes to linking to the Csound library on Windows. The easiest way is to link directly to the csound32.dll.5.2 library that ships with Csound5. This can be done by passing the following "linker option" to your command line:

```
-lcsound
```

If linking using the above command results in linker errors try renaming csound32.dll.5.2 to *csound.lib,* this should resolve any issues with linking.

The other option for building Csound host API applications involves manually creating an import library. If you built Csound from source then you will have already created this file, otherwise you will need to create an import library from the Csound dynamic link library, i.e., *csound32.dll.5.2.* To create an import library on Windows you will need to install pexports from http://www.emmestech.com/software/cygwin/pexports-0.43/download_pexports.html
I recommend putting the pexports.exe executable into your MinGW bin folder so that you don't have to add any new directories to the system path. Once you've installed pexports, you can run the following command from the Csound bin folder.

```
pexports -o csound32.dll.5.1 > csound32.def
```

The newly created *csound32.def* must now be edited in order to remove the first line that seems to cause an error when compiling. Once the first line of csound32.def has been removed it should be saved. The *libcsound32.a* import library can then be built using the following command:

```
dlltool --dllname=csound32.dll.5.1 --input-def=csound32.def --
outputlib=libcsound32.a
```

To create an import lib from the *csnd.dll* file you can simply rename it *csnd.a* or follow the instruction provided above and simply pass csnd to the linker like this:

```
-lcsnd
```

Whatever option you choose to use you will need to provide the path to the location of the Csound library to your compiler.

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. The second method is easier, and uses Scons.

---

1 Depending on the version of Csound you installed this library may be named csound32.dll.5.2 or csound64.dll.5.2. For the purposes of simplifying this guide I will refer only to csound32.dll.5.2. The same approaches and techniques are used regardless of which version of the library you install. In future releases of Csound this library name may change again but it will always have the same format, i.e, csound-version-.dll-api version.

More details on using Scons is 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 Windows 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 paragraph above. Also ensure that you specify the correct path to the folder containing the Csound import libraries. In the following examples I've renamed csound32.dll.5.2 with csound.lib so that I can link directly to the Csound library by using *-lcsound*. On Windows machines this is usually *C:/Program Files/Csound/bin.* As mentioned above be careful if copying and pasting these command as some compilers have problems with some `Courier New` typeface punctuation marks.

```
gcc example1.c -o example1.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

gcc example2.c -o example2.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

gcc example3.c -o example3.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

gcc example4.c -o example4.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

gcc example5.c -o example5.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

gcc example6.c -o example6.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

g++ example7.cpp -o example7.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

g++ example8.cpp -o example8.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

g++ example9.cpp -o example9.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd

g++ example10.c -o example10.exe -I"PATH_TO_CSOUND_FOLDER_CALLED_H”
```

```
-I"PATH_TO_CSOUND_FOLDER_CALLED_INTERFACES" -
L”PATH_TO_FOLDER_CONTAINING_CSOUND_LIBS” -lcsound -lcsnd
```

Building the Command-line Application: 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 paths to point towards the *H* and *interfaces* folders and the library paths so it points to 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! In particular they need to point
# to where the Csound folders 'H' and 'interfaces' are located. Anywhere you
# see a path you need to update for your system

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']})
      env.Append(LIBPATH = ['C:/Program Files/Csound/bin'])
      env.Append(CPPPATH = ['C:/Program Files/Csound/H', 'C:/Program
Files/Csound/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 = ['/Users/rorywalsh/SourceCode/Csound5.10.1/H',
'/usr/include', '/Users/rorywalsh/SourceCode/Csound5.10.1/interfaces'])
      env.Append(LIBPATH = ['/usr/loca/lib'])

#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 = ['csound','csnd'])

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"

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"

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. Provided with this guide are two Sconstruct files, one for building the command line applications and one for building the GUI applications. In order to instruct Scons to build with a particular file we use the `-f` flag to explicitly specify a file. For example:

```
$ scons -f SConstructCommandLine
```

This will instruct Scons to use the SConstructCommandLine script to build all the command line examples contains 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 examples you will need to download and install the wxWidgets library. On Windows the easiest thing to do is to download **wxPack** that can be downloaded from here: http://wxpack.sourceforge.net. This will provide you with all the source and precompiled binaries that you need to build these examples. Alternatively you can build the library yourself. Download the source from www.wxWidgets.org. Then using MinGW and MSYS 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 on Windows is with Scons. This is because of the number of libraries you need to link against. To build run Scons with the *SconstrucGUI* script:

```
$ scons -f SconstructGUI
```

As you can see from looking at the commands 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 your MSYS shell. Don't forget to update the paths to reflect your system:

```
g++ helloworld.cpp -o HelloWorld.exe -I"C:\Program Files\Dev-Cpp\include" -
I"C:\MyDocuments\SourceCode\Csound5\csound5\H" -
I"C:\MyDocuments\SourceCode\Csound5\csound5\interfaces" -I"C:\Program
Files\Dev-Cpp\include\common" -L"C:\Program Files\Dev-Cpp\lib" -
L"C:\MyDocuments\SourceCode\Csound5\csound5" -lcsound32 -lcsnd -lwxmsw28 -
lwxmsw28_gl -lwxtiff -lwxjpeg -lwxpng -lwxzlib -lwxregex -lwxexpat -lkernel32
-luser32 -lgdi32 -lcomdlg32 -lwinspool -lwinmm -lshell32 -lcomctl32 -lole32 -
loleaut32 -luuid -lrpcrt4 -ladvapi32 -lwsock32 -lodbc32 -lopengl32

g++ GUIexample1.cpp -o GUIexample1.exe -I"C:\Program Files\Dev-Cpp\include" -
I"C:\MyDocuments\SourceCode\Csound5\csound5\H" -
I"C:\MyDocuments\SourceCode\Csound5\csound5\interfaces" -I"C:\Program
Files\Dev-Cpp\include\common" -L"C:\Program Files\Dev-Cpp\lib" -
L"C:\MyDocuments\SourceCode\Csound5\csound5" -lcsound32 -lcsnd -lwxmsw28 -
lwxmsw28_gl -lwxtiff -lwxjpeg -lwxpng -lwxzlib -lwxregex -lwxexpat -lkernel32
-luser32 -lgdi32 -lcomdlg32 -lwinspool -lwinmm -lshell32 -lcomctl32 -lole32 -
loleaut32 -luuid -lrpcrt4 -ladvapi32 -lwsock32 -lodbc32 -lopengl32

g++ GranulatorV1.cpp -o GranulatorV1.exe -I"C:\Program Files\Dev-Cpp\include"
-I"C:\MyDocuments\SourceCode\Csound5\csound5\H" -
I"C:\MyDocuments\SourceCode\Csound5\csound5\interfaces" -I"C:\Program
Files\Dev-Cpp\include\common" -L"C:\Program Files\Dev-Cpp\lib" -
L"C:\MyDocuments\SourceCode\Csound5\csound5" -lcsound32 -lcsnd -lwxmsw28 -
lwxmsw28_gl -lwxtiff -lwxjpeg -lwxpng -lwxzlib -lwxregex -lwxexpat -lkernel32
-luser32 -lgdi32 -lcomdlg32 -lwinspool -lwinmm -lshell32 -lcomctl32 -lole32 -
loleaut32 -luuid -lrpcrt4 -ladvapi32 -lwsock32 -lodbc32 -lopengl32

g++ GranulatorV2.cpp -o GranulatorV2.exe -I"C:\Program Files\Dev-Cpp\include"
-I"C:\MyDocuments\SourceCode\Csound5\csound5\H" -
I"C:\MyDocuments\SourceCode\Csound5\csound5\interfaces" -I"C:\Program
Files\Dev-Cpp\include\common" -L"C:\Program Files\Dev-Cpp\lib" -
L"C:\MyDocuments\SourceCode\Csound5\csound5" -lcsound32 -lcsnd -lwxmsw28 -
lwxmsw28_gl -lwxtiff -lwxjpeg -lwxpng -lwxzlib -lwxregex -lwxexpat -lkernel32
-luser32 -lgdi32 -lcomdlg32 -lwinspool -lwinmm -lshell32 -lcomctl32 -lole32 -
loleaut32 -luuid -lrpcrt4 -ladvapi32 -lwsock32 -lodbc32 -lopengl32
```

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 Press 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 mentioned above, pass a sound file to the soundin opcode in granulatorV2.csd. Go to **File -> Start** to begin processing audio.