Preliminary Manual for the Audio I/O libraries

Author: Wolfgang Büscher (DL4YHF)
Date: 2014-02-22 (YYYY-MM-DD)
Original file location: cbproj/AudioIO/manual/AudioIO_Manual.odt
Website, download: Used to be at www.qsl.net/dl4yhf/AudioIO/index.html .

Contents

1 Purpose

The "Audio-I/O" system described here shall connect applications which exchange uncompressed audio streams in real time, for example...

- control programs for special audio sources (like software defined radios),
- audio-processing applications (not just plugins)
- audio-streaming applications (which send audio to a web radio, etc).

Some features of AudioIO :

- it is an open-source project (with a LGPL / BSD-like license, see next chapter)
- it shall be kept as simple as possible
- it shall only connect audio sources and -destinations, but not do any audio processing
- it doesn't occupy the soundcard, it doesn't receive anything from the soundcard, and it doesn't send anything to the soundcard (at least, not directly !)
- it does not rely on proprietary drivers, kernel streaming, virtual audio cables, etc
- it's free, and will ever be

The first application of the Audio-I/O-Plugin was in fact to build a bridge between Spectrum Lab (see www.qsl.net/dl4yhf/spectra1.html) and Nullsoft's Winamp (preferrably V2.90), for the purpose of sending a filtered audio stream from a VLF receiver to an Icecast server.

The document which you are currently reading focuses on the installation, operation, and (forfellow programmers) the internal function of the Audio-I/O-DLL(s), in the hope that other authors of free software implement a compatible interface in their software, too. For this purpose, the complete sourcecodes and a small demo program is contained in the packed archive.

2 Disclaimer, Copyright, License, Trademarks...

TRADEMARKS AND REGISTERED TRADEMARKS

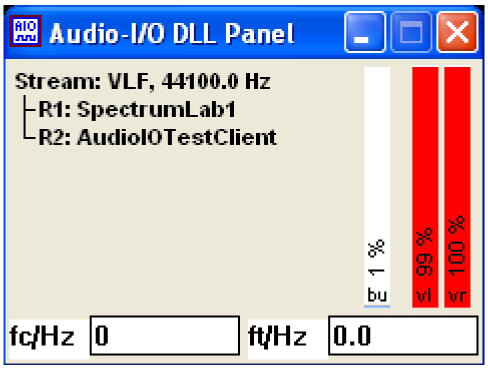
3 The library „in_AudioIO.dll“

3.1 Using „in_AudioIO.dll“ to distribute audio streams

This incarnation of an audio-I/O-library can be used to distribute uncompressed audio streams to multiple readers, for all windows programs *running on the same PC* with a compatible interface to the DLL. The program 'Spectrum Lab' (more on that in the following chapters) is one example for such a program: It can act as a stream source („writer“) as well as a destination („stream reader“). The once-famous audio player named Winamp (now owned by AOL) can also be a stream reader.

Audio processing applications which support the ASIO protocol (© Steinberg GmbH) can also use this Audio-I/O library, because if necessary it can 'emulate' an ASIO-compatible audio input device. This way, uncompressed audio can be routed from one application to another without the need for a 'virtual audio cable' or similar. More on the ASIO emulation in a later chapter.

This DLL has a crude control panel, which can be opened from any program which has loaded the DLL. For example in Spectrum Lab, if this DLL is used as the audio input (instead of a soundcard), you can open the following control panel from Spectrum Lab's main menu through ***Options*** … ***Show Control Panel for Audio Input DLL*** :


Audio-I/O DLL Panel

Stream: VLF, 44100.0 Hz
├R1: SpectrumLab1
└R2: AudioIOTestClient

On the default view of this control panel, there's a list of all currently active audio streams (which „use this DLL“), along with a list of all „stream readers“ which currently use to those streams for input.

Example (from the screenshot above): Stream „VLF“ with 44100 Samples/second feeds two readers:

- R1 (=the first reader) is an application which identified itself as „SpectrumLab1“ (when connecting to the stream);
- R2 (=the second reader) is an application which identifies itself as „AudioIOTestClient“ (this is actually a small console application which is contained in the sourcecode archive of the 'demo Audio-I/O-DLL').

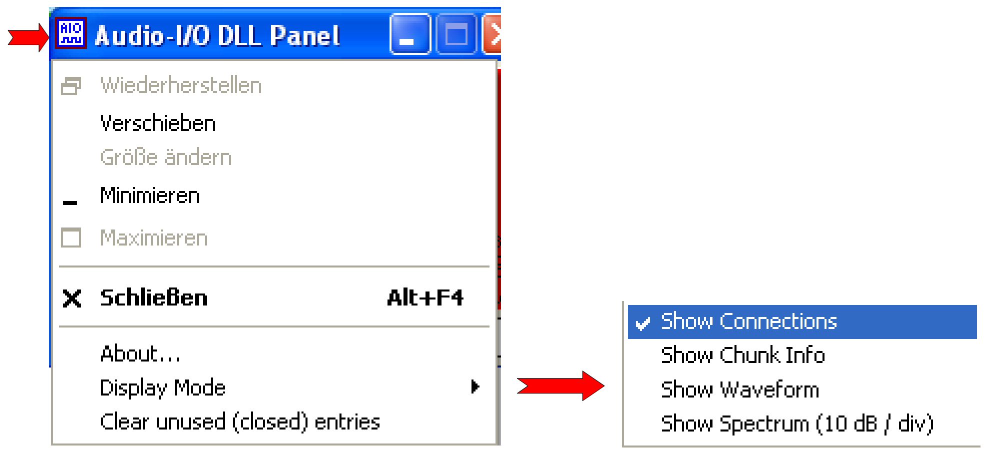
On the right side of the control panel, three (or more) vertical indicator bars are displayed:

- „bu“ shows the audio stream buffer usage in percent. Above 90 % means „the buffer will overflow very soon, because one or more audio stream readers cannot keep up the pace“.
- „vl“ is the currently measured peak volume of the left audio channel (or the „I“ -inphase- channel of an I/Q stream from a software-defined radio)
- „vr“ is the currently measured peak volume of the right audio channel (or „Q“ - quadrature- channel of an I/Q stream from a software-defined radio)

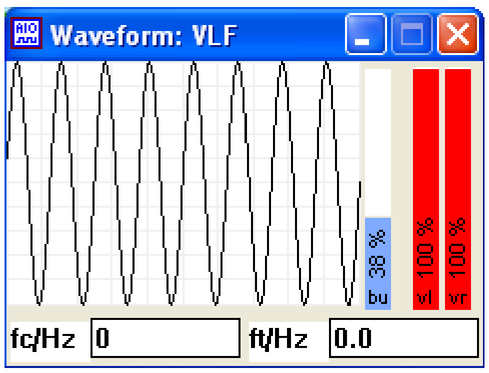
Besides the „Connections“ display shown in the screenshot above, this special audio I/O library has a few other display modes, which are described in the next chapters.

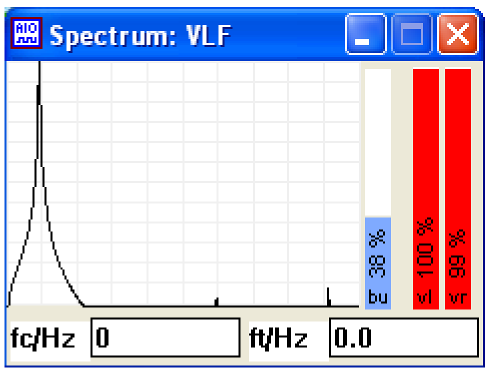
3.2 Control panel of the „in_AudioIO.dll“

To keep the DLL's own control panel as small as possible, it doesn't have a standard menu - just a small popup menu. To open that menu, click on the 'application icon' (formerly know as 'window icon', here „AIO“ symbol) in the ***upper left corner***, or press ALT+spacebar (which opens the current window's „window menu“, aka „system menu“). The items at the end of the menu are specific to this DLL (here: in_AudioIO.dll, there may be more entries in future versions).



The ***Display Mode*** menu can be used to switch from the default view („Show Connections“) to details about the currently selected audio stream, the waveform (shown below on the left side), and the momentary spectum (on the right).





Note: These displays are not intended as a replacement for a full-grown spectrum analyser, but as a quick tool to check for proper input levels („not too weak, but no risk overload“). The graph should not touch the upper or lower end of the 'waveform' display (which means the maximum input voltage of the analog/digital converter is reached, and the signal may get distorted by overloading / clipping). On the other hand, the signal should not look like an „almost flat line“ (which indicates that the dynamic range is wasted, because only a few bits of your 16- or 24-bit A/D converter are actually 'busy').

Closing the DLL's control panel doesn't stop the audio processing - it just hides the window.

3.3 *Using „in_AudioIO.dll" as **input or output device** in Spectrum Lab*

The following steps were used in Spectrum Lab V2.79, tested under Windows XP. Under later windows versions, there may be OS-specific restrictions for the directories where the audio-I/O-DLL may be copied.

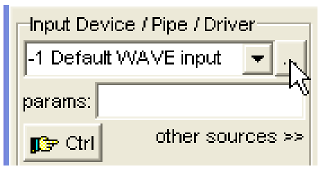
On the author's PC, the DLL was located at C:\cbproj\AudioIO\in_AudioIO.dll (because it was **not** intended to be used as a Winamp plugin, the directory path is not important).

For testing purposes, two instances of Spectrum Lab (SL) were launched on the same PC:

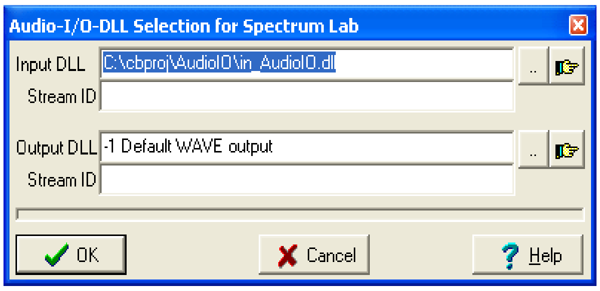
- The **first** instance used in_AudioIO.dll as its audio output device,
  i.e. Spectrum Lab was configured to produce an audio stream which it sent to the DLL
- The **second** instance used in_AudioIO.dll as its audio input device (=source),
  i.e. this instance was configured to read an audio stream from the DLL

Here are the steps to select and configure in_Audio.dll directly as **input device** for Spectrum Lab (load 'directly', i.e. not as an emulated ASIO device).

1. In SL's main menu, select **Options**, 'Audio settings, I/O device selection'
2. Under '**Input device** / Pipe / Driver', click on the three-dotted button ('…') to open a file selector :



3. In the file selector (which looks very different depending on the OS), find your way (full directory path) to in_AudioIO.dll . Finding a real directories can be painful these days: Depending on the OS, and the installed language, you may have to click on 'Arbeitsplatz', 'Computer', or whatever-they-call-it-now to see local disk drive „C:" (i.e. the drive onto which you have copied in_AudioIO.dll)
   If not done automatically, set the 'file type' to 'Dynamic Link Libraries (*.dll).
4. Close the file selector dialog (e.g. click 'OPEN', depends on the OS).
   SL may prompt you to 'Enter additional parameters'. Leave that field empty, click 'OK'.
   This should take you back to the 'Input Device / Pipe / Driver' panel in SL shown above.
   Leave the field 'params:' empty. This field may contain extra parameters passed via command line string (when loading the DLL from SL) which are not necessary for this kind of DLL.
5. Click on the 'Ctrl' button (Ctrl) to open the Audio-I/O-DLL control dialog in SpecLab
6. SL shows an extra dialog titled 'Audio I/O-DLL Selection for Spectrum Lab'.


Audio-I/O-DLL Selection for Spectrum Lab

Input DLL: C:\cbproj\AudioIO\in_AudioIO.dll
Stream ID:
Output DLL: -1 Default WAVE output
Stream ID:

The audio '**input**' (i.e. audio read by Spectrum Lab from DLL instead of the soundcard's A/D converter) is configured in the upper part of that panel, labelled 'Input DLL'. In this case, you don't need to provide a 'stream ID'.

The audio '**output**' (i.e. audio sent from Spectrum Lab to the DLL, or to the soundcard's D/A converter) is configured in the lower part of that panel, labelled 'Output DLL'. Because in this case, Spectrum Lab acts as the 'audio producer' (source), it must provide a proper and unique name for the stream. Enter that name in the 'Stream ID' field. The audio-I/O-DLL will display the stream-ID on its own control panel. For example, let's call the stream 'VLF':

Input DLL : C:\cbproj\AudioIO\in_AudioIO.dll (**for the audio-consuming instance**)
StreamID : VLF (just an example, used throughout this document)

Output DLL : C:\cbproj\AudioIO\in_AudioIO.dll (**for the audio-producing instance**)
StreamID : VLF

If you need to modify the stream-ID later, you can open the 'Audio I/O-DLL Selection' dialog later by clicking the button 'hand pointing right' (☞ Ctrl) on the 'Audio I/O' tab.

Notes:

- The DLL's own control panel can be opened from Spectrum Lab's main menu through ***Options … Show Control Panel for Audio Input DLL***
- As long as there is not a single 'reader' for the audio stream sent to the DLL, the DLL signals a problem to the DLL host. In the sample setup described above, this happens as long as only the 'audio producer' (first instance of Spectrum Lab) is running, but not the 'audio consumer' (second instance).
  Spectrum Lab may then indicate a problem with the output device[1] in its Circuit Window, by a red coloured 'output' block at the right side of the display.

1 The 'output device' is usually labelled 'DAC' = digital/analog converter. But when the output is routed to an audio-I/O-DLL, the label changes to 'AudioIO'.

*3.4 Using „in_AudioIO.dll“ **as a bridge** between Spectrum Lab and WSQ*

(WSQ = Weak Signal QSO mode, developed by ZL2AFP, adapted by DL4YHF to accept any of the Audio-I/O-DLLs as an *input* device. Spectrum Lab used as pre-processor, i.e. audio source for WSQ. A modified version of WSQ, which can load Audio-I/O-DLLs as input device, can be downloaded from www.qsl.net/dl4yhf/WSQ/index.html)

On the Spectrum Lab side: Select „in_AudioIO.dll“ as an **output** device (from SL's point of view). Details in the previous chapter.

On the WSQ side: Select „in_AudioIO.dll“ as an **input** device (from WSQ's point of view). Details will be in the WSQ manual, hopefully soon .. (2014-02-22: Still in the experimental state).

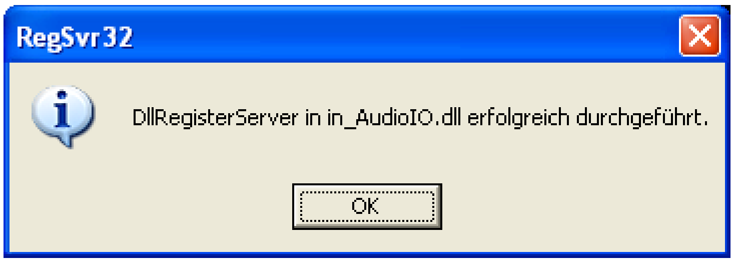
4 Installation of „in_AudioIO.dll“ as an ASIO device

At least the library 'in_AudioIO.dll' can emulate a soundcard with an ASIO driver. This way, the DLL can be used as a bridge between two audio processing applications (if they support ASIO): Application A may use the DLL as the output device, while applications B (or even more) can use the DLL as ASIO input device, to receive (read) the audio stream which was produced by application A.

To register („install“) the DLL as an ASIO driver, invoke the batchfile 'register.bat' in the same directory where the DLL file is located, too. If the batchfile is missing, you can alternatively invoke this command through the windows 'Start' menu (Start->Run, or in German / auf Deutsch: Start->Ausführen):

```
regsvr32.exe in_AudioIO.dll
```

(actually, this is the commandline contained in the file 'register.bat'). Windows should inform you about a successful registration with a message box similar like the following (here: on a German PC):


RegSvr32

DllRegisterServer in in_AudioIO.dll erfolgreich durchgeführt.

After this, audio-processing applications like Spectrum Lab, I2PHD's Winrad, and possibly some others will show the following new entry in the list of ASIO devices:

ASIO/AudioIO-Gateway

If, for some reason, you need to unregister („uninstall“) the ASIO driver functionality again, invoke the batchfile 'unregister.bat', or run the following command:

```
regsvr32.exe -u in_AudioIO.dll
```

5 Installation of „in_AudioIO.dll“ as a Winamp input plugin

From a user's point of view, there is no installation required.
To use the AudioIO DLL *in connection with Winamp*, simply copy the file "in_AudioIO.dll" into Winamp's plugin directory.

- On a PC with english windows installlation, this will typically be
  - `c:\Program Files\Winamp\Plugins`
- On a german PC, it will be
  - `c:\Programme\Winamp\Plugins`
- On an italian PC, it will be something like
  - `c:\Programmi\Winamp\Plugins`

Note:

Do not copy the file in_AudioIO.dll into the directory of the application which uses it (like your favourite SDR software), because it is important that all applications which use the DLL load it from the same file. If you have multiple copies of in_AudioIO.dll on your harddisk, Winamp may load a different copy than the application. In that case, the audio exchange between both applications won't work. Programmers please read the notes on AIO_LoadDLL() .

To check if the input plugin has successfully been "installed" (well, copied to) Winamp...

- start Winamp
- press CTRL-P (or select "Options".."Preferences" in Winamp's menu. To open the menu, click on the sine-wave-like icon in Winamp's upper left corner)
- select "Plugins"..."Input" in the tree view on the left
- There must be an item called "Input from Audio-IO vX.Y [in_AudioIO.dll]" in the list of Input plug-ins. You can click on that item or select "Configure" when the item is selected, but that's not required this time.
- Close the window titled "Winamp Preferences" again.

6 Operation as an input-plugin for Winamp

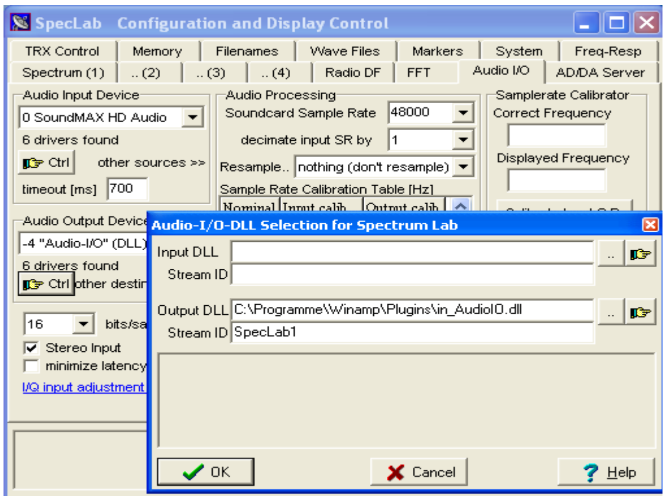
After installing the library „in_AudioIO.dll“ as a Winamp plugin (as explained in the previous chapter), we will use Winamp to receive an audio stream from the plugin now.
For a start, we use the plugin's built-in test tone generator (which even works if we don't have an external application producing an audio stream).

- In Winamp, press CTRL-L (or select "Play"..."URL...") in Winamp's menu.
  (if the menu isn't visible, click on the sinewave icon in Winamp's upper left corner)
- In the box titled "Open URL", enter
  `tone://1000`
  and press enter (or click "Open").
- You should now hear a clean 1000 Hz audio tone, because that is what the plugin will send to Winamp (when Winamp asks it to open the "URL" tone://1000).

How does this work, and what is this "tone://1000"-thing ?
To Winamp, it looks like a URL (Unified Resource Locator, but forget about that for a moment). Here, it is just a pseudo-URL, because neither "tone" (nor "audi", see below) are valid internet protocols. But Winamp doesn't care if it's a valid URI, URL, filename, or whatever. Instead, Winamp asks all installed input-plugins if one of them knows what to do with a URL beginning with "tone://" (or "audi://", or whatever). The AudioIO-plugin (at least, in_AudioIO.dll) will jump in, saying "hello Winamp, yes I can open that address". So, in the next step, it will be asked to open that URL.
The plugin then parses the first part of the address (which isn't a URL at all) and decide what to do. Up to now, the following tokens (which would be the protocol in a URL, like http) have been implemented in the Audio-I/O plugin :

- **`tone://`**<frequency in Hz>
  produces a test tone of the specified frequency
- **`audi://`**<optional name of an audio source>
  tells the plugin to listen for the specified audio source
  (in Latin language: "audi !" = "listen !" , because for simplicity the author needed a 4-letter word)

Example: Spectrum Lab provides a stream named 'SpecLab1' through the audio-I/O-DLL, which Winamp shall read (there are different ways to achieve this, see the [SL documentation]()).



Audio Input Device
0 SoundMAX HD Audio
6 drivers found
timeout [ms] 700

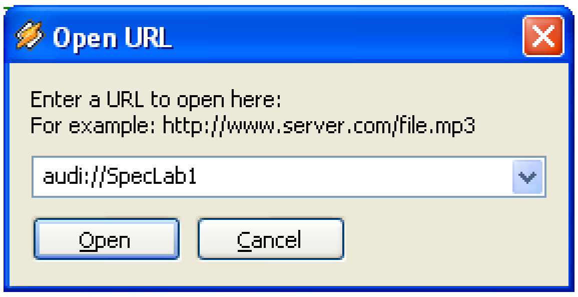
Audio Processing
Soundcard Sample Rate 48000
decimate input SR by 1
Resample.. nothing (don't resample)
Sample Rate Calibration Table [Hz]

Samplerate Calibrator
Correct Frequency
Displayed Frequency

Audio Output Device
-4 "Audio-I/O" (DLL)
6 drivers found
16 bits/sa
☑ Stereo Input
☐ minimize latency

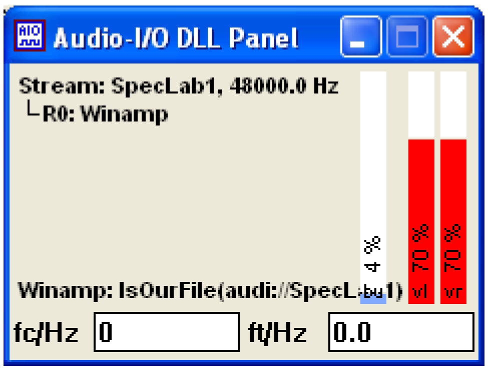
Audio-I/O-DLL Selection for Spectrum Lab
Input DLL
Stream ID
Output DLL C:\Programme\Winamp\Plugins\in_AudioIO.dll
Stream ID SpecLab1

In Winamp, click on the sinewave symbol (upper left corner), select 'Play', 'URL', and ...


Open URL
Enter a URL to open here:
For example: http://www.server.com/file.mp3
audi://SpecLab1
Open
Cancel

… enter the pseudo-URL 'audi://', followed by the name of the audio stream, in the dialog box as shown above.

You can now check the 'audio connections' realized by the audio-I/O-DLL (in this case, in_AudiIO.dll) by opening the DLL's own control panel. This can be achieved through winamp's menu (Options .. Preferences .. Plugins; select 'Input from Audio-IO and click 'Configure') or through Spectrum Lab's „Audio-I/O-DLL Selection“ dialog shown further above (click on the small buttons with the 'hand pointing right' : Ctrl


Stream: SpecLab1, 48000.0 Hz
└R0: Winamp

Winamp: IsOurFile(audi://SpecLab1)

Details about the audio-I/O-DLL's control panel, and the different display modes, are in another chapter.

7 Software Description

The following chapters are only intended for software developers, who'd like to use the AudioIO principle in their own applications.
If your plan is to simply use the existing AudioIO library, don't read this chapter. The previous chapter already told you all you need to know about using the Audio-I/O system.
If you want to write your own AudioIO-compatible library, read the following chapters, too.

7.1 How the Audio-I/O-libraries work internally

The module AudioIO.c dynamically loads an AudioIO-PLUGIN-DLL (for example in_AudioIO.dll) The DLL itself contains a circular audio buffer IN SHARED MEMORY . Winamp can also load the same DLL (from the same location !!), and other programs can use the same DLL at the same time

(if they support AudioIO, or ASIO, or can use the Winamp plugin interface) .

So both applications ("processes") can exchange audio streams without the need for multiple soundcards, virtual audio cables, shared files, windows messages, etc.

For details about shared memory (under windows), look for an article named „Creating Named Shared Memory“. In 2011, that article was available online at :
http://msdn.microsoft.com/en-us/library/aa366551%28v=vs.85%29.aspx

At the time of this writing, no equivalent of such globally shared memory (shared between processes) for the Linux environment was known to the author, but surely such a mechanism does exist.

Functions used to access the shared memory (at least in in_AudioIO.dll) are:

- CreateFileMapping (with INVALID_FILE_HANDLE, because we don't want to use a 'real' file for this, but shared ***memory***)
- MapViewOfFile (to retrieve a pointer into the shared memory area, which is *only valid for the callng process*)
- UnmapViewOfFile (counterpart to MapViewOfFile)
- CloseHandle(to free the handle returned by CreateFileMapping)

Most audio I/O libraries (at least those based upon in_AudioIO.dll) save their configuration in an old-fashioned ini file. This used to be saved in the same directory as the DLL itself, but thanks to some stupid restriction in newer windows versions, this is no longer allowed (when the program which hosts the DLL is run with normal 'user' priviledges). So, since 2011, the configuration of the audio I/O DLL is now saved in the windows directory, filename **AUDIO_IO_DLLS.INI** .

Here is an excerpt of that file (written by the DLL, you don't need to edit it yourself):

```
[AudioIOControlPanel]
x1=10              <<<  when visible,
y1=10              <<<  this is the position of the control panel on the desktop
DisplayMode=0
```

```
[About]
i1=Configuration file for DL4YHF's Audio-I/O-DLL, compiled Sep 14 2011
i2=c:\cbproj\AudioIO\in_AudioIO.dll  <<< path to the DLL which wrote this file

[AudioSource1]                       <<< config section for the first audio source / stream
StreamID=VLF
Description=Source description
iSampleRate=44100
nChannelsPerSamplePoint=2
iDisableOutput=0
iDayLastUsed=15231
```

Especially the 'About' section helps with troubleshooting: It contains the complete file path to the DLL itself. Whenever the last instance of the DLL is unloaded from memory, it refreshes this info.. so in case of doubt you can find out where the DLL is located (in winamp's plugin folder, or somewhere else...).

Note: A host (i.e. „program which wants to load this DLL“) should not rely on the presence of this INI file, because the INI file only exists if the DLL has been loaded at least once !

7.2 Sourcecode modules

AudioIO.H

contains all function prototypes, data types, error codes and similar which are common to all software modules in the "Audio-I/O" package. There are exotic data types like INT16 and INT32 in this header (which replace the standard C types "int" and "long"), to make sure the data types inside the DLL (compiled with DevCpp, for example) and the data types inside the interface module (compiled with Borland C++, for example) are compatible.
So AudioIO.H also provides a "compatibility layer" between the application and the code within the AudioIO DLL .

AudioIO.C

is just an interface between the AudioIO-DLL and the application. It dynamically loads the DLL into memory on request, to avoid the issues with compiler-specific name mangling, the need for import libraries (*.lib), etc .

aio2winamp.C

is the main sourcecode of the first implementation of an AudioIO-compatible DLL. In this case, it is also an "input plugin" for Nullsoft's Winamp. Compiled with DevCpp (V4.9.9.2), using the project file "in_AudioIO_DevCpp.dev" to produce the file "in_AudioIO.dll" .

The DLL prefix "in_" is dictated by Winamp to recognize it as an input-plugin.

ASIO\ASIO_driver_in_C.c

is the skeleton of a 'COM' object (Microsoft Coponent Object Model) written in „plain C", to avoid having to use megatons of bulky include files. We need this COM-stuff because an ASIO driver, when implemented on a windows platform, is in fact a COM object.
Note that this file is NOT one of the files in Steinberg's ASIO SDK, and it's also NOT derived on Steinberg's own 'ASIO driver example' (because to compile Steinberg's ASIO sample driver, we would need a microsoft „visual-something" compiler).

ASIO\AsioDriverThiscalls_BCB.c

is required because very unfortunately, Steinberg uses Microsoft's proprietrary „thiscall" calling convention in „their" ASIO driver API. This module implements a few interface functions, which are called using the „thiscall" calling convention (which means the 'THIS' pointer is passed in the ECX register). Most of the code in this functions do a „stdcall" (standard calling convention, which is supported by any C compiler, not just Microsoft's) to the ASIO driver functions implemented somewhere else. In other words, this module contains just the interface code, written in Borland's easy-to-understand inline assembly (thus the suffix „BCB" in the suffix of the filename).

For details on the „thiscall" problem with ASIO drivers, read the article by Ross Bencina titled „IASIOThiscallResolver". But note that here, in the implementation of an ASIO driver (not an ASIO host), the „thiscall" must be resolved the other way around: The CALLER (ASIO host) uses „thiscall", which the CALLEE (ASIO driver) must convert into a standard call supported by the C compiler.

Please read the notes on Steinberg's ASIO SDK Licensing further below, if you miss the files „asio.h" and"asiosys.h" in the Audio-IO sourcecode archive ! (Thanks to Steinberg, I am not allowed to include them here).

AudioIO_Test_Client.c

is a miniature test application which sends a few seconds of data to Winamp, using the "in_AudioIO" plugin. How to compile and run this program is described in the next chapter.

If you want to modify and/or recompile the project with ASIO functionality, you will need to download the ASIO SDK from Steinberg. Steinberg's licensing policy doesn't allow me (or anyone else) to include „their" files here. Quoted from Steinberg's ASIO Licensing Agreement, which you will have to sign before you can download their ASIO-SDK:

> The Licensee has no permission to sell, license, give-away and/or distribute the **Licensed Software Developer Kit** or parts of it in any way, on any medium, including the Internet, to any other person, …

Btw it's amazing to see how often this License agreement has been violated. But, being a good citizen, sign up at Steinberg to get your own, licensed copy of the ASIO SDK (which, by the way, is completely free). In 2011, the ASIO SDK was available at

http://www.steinberg.net/en/company/developer.html

If you prefer *not* to include ASIO support when compiling the audio-I/O demo project, you can remove the ASIO support by modifying the following line in SWITCHES.H :

To compile in_AudioIO.dll with ASIO support:

```
    #define SWI_ASIO_SUPPORTED    1   /* Support ASIO audio drivers ?          */
```

To compile in_AudioIO.dll without ASIO support (and without the 'Steinberg' ASIO files) :

```
    #define SWI_ASIO_SUPPORTED    0   /* Support ASIO audio drivers ?  0=no  */
```

In a similar way, you can also remove the WINAMP support. The include file SWITCHES.H is located in a special directory, which is specified as include directory for the C compiler somewhere in the project file. If you use Borland C++ Builder V6 on a german PC, here's how to modify the project-specific include-paths:
„Projekt“..“Optionen“..“Verzeichnisse/Bedingungen“..“Ordner“..“Include-Pfad“.
(with an english installation of BCB V6, this would be something like
„Project“..“Options“..“Directories/Conditionals“..“Directories“..“Include Path“).

There used to be a project file for GNU C (GCC, actually MinGW / DevCpp) but since Borland's inline assembly language is MUCH easier to use and understand than GCC's dreadful „double-quoted“ inline assembler syntax, the Dev-Cpp project (in_AudioIO_DevCpp.dev) will most likely be outdated.

7.3 The AudioIO test application (written in C)

To compile and run this application, you need -of course- a C- or C++ compiler. It has been written and tested with DevCpp (which uses the MinGW compiler), or Borland C++Builder. You can use it as a testbed for your own audio-I/O-libraries (for example, as an interface between an 'exotic' hardware and programs like Spectrum Lab), or as a starter for your own application which uses these libraries (i.e. a program whixh also „hosts“ the DLL).

The audio-I/O test application (AudioIO_Test_Client.exe or AudioIO_Test_Client_BCB4.exe) is a command-line driven, and partly keyboard controlled ***console application***. Command line arguments are:

- the first argument is the filename of the DLL to be tested (including the directory path)
- /c Show control panel
- /i Test input. Optionally, a stream name can be specified (see example below)
- /o Test output. Optionally, a stream name can be specified (see example below)

Both in- and output can be tested at the same time. In that case, the tester generates a stream of audio samples, sends it to the DLL (as „audio output“), and reads it back from the DLL (as „audio input“).
Example:

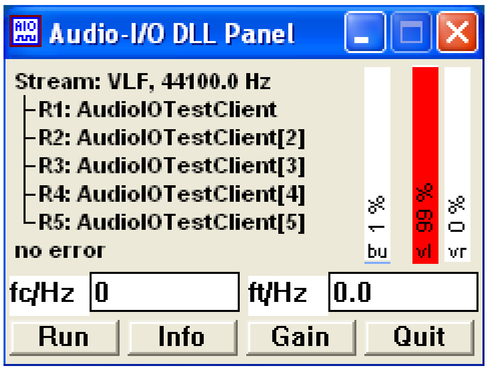
AudioIO_Test_Client.exe c:\cbproj\AudioIO\in_AudioIO_BCB4.dll /c /o=VLF /i=VLF

(loads the specified DLL, opens the control panel, opens a stream named „VLF“ for output, opens a stream named „VLF“ for input, and -in a loop- writes to the output, and reads from the input)

While the DLL-test runs, you can control it through the keyboard (remember, the „test client“ itself is a console application, not a fancy windows program):

- 'o' (lower case letter 'O') closes the audio-output (this will cause the DLL's audio buffer to „run empty“ after a few seconds, if the input remains open during this time)
- 'O' opens the audio-output again (using the stream name from the command line)
- 'i' closes the audio-input (this will cause the DLL's audio buffer to be filled up to the limit, if the output remains open during this time)
- 'I' opens the audio-input again (using the stream name from the command line)

The real fun starts if you don't load the audio-DLL from a single instance. Use multiple instances of the test client: One to send the stream, and multiple other instances to read the stream:



```
Stream: VLF, 44100.0 Hz
 ├R1: AudioIOTestClient
 ├R2: AudioIOTestClient[2]
 ├R3: AudioIOTestClient[3]
 ├R4: AudioIOTestClient[4]
 └R5: AudioIOTestClient[5]
no error
fc/Hz 0        ft/Hz 0.0
```

The sourcecode ("AudioIO_Test_Client.c") is heavily commented so we don't need to go into details. If you want to compile or modifiy it yourself :

- Download and install DevCpp (precisely: DevC++ 4.9.9.2) if not done yet :
- www.bloodshed.net
- If you also want to recompile the DLL(s), you may need to update the MinGW "binutils": www.mingw.org
- Unpack the contents of the AudioIO archive into a directory of your choice, including all subdirectories (if any) contained in the archive. The sourcecode archive used to be at www.qsl.net/dl4yhf/AudioIO/index.html (by the time of this writing, in 2011).

If the DLL shall act as an input-plugin for Winamp, copy the file "in_AudioIO.dll" into the winamp plugin directory (because we'll use Winamp as the output device). After that, you start Winamp as explained in chapter Operation . In the box titled "Open URL", simply enter audi:// .

< To be continued >

7.4 Software Details

(At the moment, these are just snippets of info which may, or may not, be helpful for other developers. If you only need to use the Audio-I/O-libraries, ignore the following chapters for your own peace of mind ;-)

7.4.1 Winamp specific details

If you want to roll your own "Audio-I/O compatible" plugin for Winamp (instead of using the already existing library in_AudioIO.dll), you will need Nullsoft's Winamp SDK . It contains all headers (like in2.h, out.h) and a few sample plugins which help you to get started.

> (Don't use or try to understand the sourcecode "aio2winamp.c" from the AudioIO source bundle, if you are not familiar with Winamp plugins. The most recent Winamp SDK may contain a more up-to-date documentation than the one I used when I wrote "my" first winamp plugin. But even in 2008, a good Winamp SDK documentation was hard to find. At least, I didn't find anything I'd call "documentation" at www.winamp.com. Even the SDK found there -winamp v5.02 SDK- seemed to be severely outdated... strange...)

From Winamp's point of view, the AudioIO-to-Winamp plugin (in_AudioIO.DLL) is just an ordinary **input plugin** like many others. There is a worker thread in it (here called WA_ExternalInputThread()), which runs in an endless loop as soon as Winamp calls the "Play()"-function through a function pointer in the In_Module structure.
Winamp's Play()-function will give us the name of the "file" it wants to play. Usually, this is the name of a disk file, but in this case, the filename tells our plugin where winamp wants to be connected ("receive the audio from"). A few examples:

- tone://1000
  generates a test signal (a sine wave) with exactly 1000 Hz instead of receiving the audio from somewhere.
- audi://
  (latin: "audi !" = "listen ! ") tells the plugin that Winamp would like to receive audio from whoever sends an audio stream to "the other end of the line".
- audi://SpecLab
  tells the plugin that Winamp would like to receive audio from an application (or, a "named

audio source") called "SpecLab" . (by the time of this writing, it was just a plan to support multiple audio sources and -destinations in a single DLL)

The worker thread in the Winamp *input* plugin periodically checks, if Winamp (or, more precisely, Winamp's currently loaded *output*-plugin) is ready to accept new data. If so, and a sufficiently large number of audio samples is present in the shared memory (which we use as FIFO-like audio buffer), the worker thread sends the data from the DLL's shared FIFO to Winamp's output-plugin. If no data are available, or the output plugin isn't able to process them at this time, the worker thread simply sleeps for 50 milliseconds before it tries again.

7.4.2 Initialisation sequence of audio sender and -receiver

In a real world, we don't know in which order the user will start the „sending“ and the „receiving“ application. Consider the following scenarios:

Scenario A: "Sender" started first, "Receiver" started last .
In this case, the receiver (for example, Winamp) will know everything it needs to know when beginning to play, because the Sender already called AIO_OpenAudioOutput(). This includes setting the sample rate and the number of channels. No problem this way.

Scenario B: "Receiver" started first, "Sender" started last .
Here, Winamp wouldn't know anything about the audio stream format, because the Sender isn't there yet. But it needs to know at least the sample rate, the number of channels, and the number of bits per sample in advance (why ? Because Winamp cannot be blocked in the call of the "Open"-function).
To solve this problem (at least, partially), the AudioIO plugin saves the last settings (sampling rate, sample format, number of channels, etc) in this configuration file:

C:\WINDOWS\AUDIO_IO_DLLS.INI

There is an extra section for the Winamp "input"-plugin with the following contents (by the time of this writing):

```
[AudioIO_to_Winamp]
iSampleRate=44100
iNumChannels=2
iBitsPerSample=16
iDisableOutput=0
```

Note: The present configuration will only be written back to the file when the *last application* which uses the DLL is terminating !