

# Preliminary Manual for the Audio I/O libraries

Author: Wolfgang Büscher (DL4YHF)

Date: 2008-03-29 (YYYY-MM-DD)

## 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 <http://freenet-homepage.de/dl4yhf/spectral1.html> ) and Nullsoft's Winamp, for the purpose of sending a filtered audio stream from a VLF receiver to an Icecast server.

A few notes about about the configuration of Spectrum Lab, the Audio-I/O interface, Winamp, Oddcast can be found here : <http://freenet-homepage.de/dl4yhf/AudioIO/index.html> .

The document which you are currently reading focuses on the installation, operation, and (for fellow 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...

Copyright (c) 2008...2065, Wolfgang Büscher (DL4YHF) .

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the author nor the name of other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### TRADEMARKS AND REGISTERED TRADEMARKS

Namings for products, that are registered trademarks, are not separately marked in these documents. The same applies to copyrighted material. Therefore the missing (tm), ®(r), or ©(c) does not implicate, that the naming is a free trade name. Furthermore the used names do not indicate patent rights or anything similar.

### 3 Installation

From a user's point of view, there is no installation required.

To use the AudioIO system *in connection with Winamp*, simply copy the file "in\_AudioIO.dll" into Winamp's plugin directory.

- On a PC with english windows installation, 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.

## 4 Operation

After installing the 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 you don't see the damned menu, click on the sine-wave-like icon in Winamp's upper left corner)
- In the box titled "Open URL", enter  
**tone://1000**  
and press enter (or click "Open").
- You should not 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 "I can open that address"). So, in the next step, it 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 know, 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)

## 5 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.

If you want to write your own AudioIO-compatible library, read the following chapters, too.

### 5.1 *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 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.

## 5.2 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.

### 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.

### 5.3 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 .

The sourcecode ("AudioIO\_Test\_Client.c") is heavily commented so we don't need to go into details. To begin from scratch...

- Download and install DevCpp (precisely: DevC++ 4.9.9.2) if not done yet :
- [www.bloodshed.net](http://www.bloodshed.net)
- If you also want to recompile the DLL(s), you may need to update the MinGW "binutils":  
[www.mingw.org](http://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 here:  
<http://freenet-homepage.de/dl4yhf/AudioIO/AudioIO.zip>

Before starting the test application, 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 >

### 5.4 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, please ignore the following chapters for your own peace of mind ;-)

#### 5.4.1 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...)

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.

## 5.4.2 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 !