# **libsndfile : Frequently Asked Questions.**

[Q1 : Do you plan to support XYZ codec in libsndfile?](#gjdgxs)

[Q2 : In version 0 the SF\_INFO struct had a pcmbitwidth field but version 1 does not. Why?](#30j0zll)

[Q3 : Compiling is really slow on MacOS X. Why?](#1fob9te)

[Q4 : When trying to compile libsndfile on Solaris I get a "bad substitution" error during linking. What can I do to fix this?](#3znysh7)

[Q5 : Why doesn't libsndfile do interleaving/de-interleaving?](#2et92p0)

[Q6 : What's the best format for storing temporary files?](#tyjcwt)

[Q7 : On Linux/Unix/MacOS X, what's the best way of detecting the presence of libsndfile?](#3dy6vkm)

[Q8 : But I just want a simple Makefile! What do I do?](#1t3h5sf)

[Q9 : How about adding the ability to write/read sound files to/from memory buffers?](#4d34og8)

[Q10 : Reading a 16 bit PCM file as normalised floats and then writing them back changes some sample values. Why?](#2s8eyo1)

[Q11 : I'm having problems with u-law encoded WAV files generated by libsndfile in Winamp. Why?](#17dp8vu)

[Q12 : I'm looking at sf\_read\*. What are items? What are frames?](#3rdcrjn)

[Q13 : Why can't libsndfile open this Sound Designer II (SD2) file?](#26in1rg)

[Q14 : I'd like to statically link libsndfile to my closed source application. Can I buy a license so that this is possible?](#lnxbz9)

[Q15 : My program is crashing during a call to a function in libsndfile. Is this a bug in libsndfile?](#35nkun2)

[Q16 : Will you accept a fix for compiling libsndfile with compiler X?](#1ksv4uv)

[Q17 : Can libsndfile read/write files from/to UNIX pipes?](#44sinio)

[Q18 : Is it possible to build a Universal Binary on Mac OS X?](#2jxsxqh)

[Q19 : I have project files for Visual Studio / XCode / Whatever. Why don't you distribute them with libsndfile?](#z337ya)

[Q20 : Why doesn't libsndfile support MP3? Lots of other Open Source projects support it!](#3j2qqm3)

[Q21 : How do I use libsndfile in a closed source or commercial program and comply with the license?](#1y810tw)

[Q22 : What versions of windows does libsndfile work on?](#4i7ojhp)

[Q23 : I'm cross compiling libsndfile for another platform. How can I run the test suite?](#2xcytpi)

## 

**Q1 : Do you plan to support XYZ codec in libsndfile?**

If source code for XYZ codec is available under a suitable license (LGPL, BSD, MIT etc) then yes, I'd like to add it.

If suitable documentation is available on how to decode and encode the format then maybe, depending on how much work is involved.

If XYZ is some proprietary codec where no source code or documentation is available then no.

So if you want support for XYZ codec, first find existing source code or documentation. If you can't find either then the answer is no.

**Q2 : In version 0 the SF\_INFO struct had a pcmbitwidth field but version 1 does not. Why?**

This was dropped for a number of reasons:

* pcmbitwidth makes little sense on compressed or floating point formats
* with the new API you really don't need to know it

As documented [here](http://www.mega-nerd.com/libsndfile/api.html#note1) there is now a well defined behaviour which ensures that no matter what the bit width of the source file, the scaling always does something sensible. This makes it safe to read 8, 16, 24 and 32 bit PCM files using sf\_read\_short() and always have the optimal behaviour.

**Q3 : Compiling is really slow on MacOS X. Why?**

When you configure and compile libsndfile, it uses the /bin/sh shell for a number of tasks (ie configure script and libtool). Older versions of OS X (10.2?) shipped a really crappy Bourne shell as /bin/sh which resulted in **really** slow compiles. Newer version of OS X ship GNU Bash as /bin/sh and this answer doesn't apply in that case.

To fix this I suggest that you install the GNU Bash shell, rename /bin/sh to /bin/sh.old and make a symlink from /bin/sh to the bash shell. Bash is designed to behave as a Bourne shell when is is called as /bin/sh.

When I did this on my iBook running MacOS X, compile times dropped from 13 minutes to 3 minutes.

**Q4 : When trying to compile libsndfile on Solaris I get a "bad substitution" error on linking. Why?**

It seems that the Solaris Bourne shell disagrees with GNU libtool.

To fix this I suggest that you install the GNU Bash shell, rename /bin/sh to /bin/sh.old and make a symlink from /bin/sh to the bash shell. Bash is designed to behave as a Bourne shell when is is called as /bin/sh.

**Q5 : Why doesn't libsndfile do interleaving/de-interleaving?**

This problem is bigger than it may seem at first.

For a stereo file, it is a pretty safe bet that a simple interleaving/de-interleaving could satisfy most users. However, for files with more than 2 channels this is unlikely to be the case. If the user has a 4 channel file and want to play that file on a stereo output sound card they either want the first 2 channels or they want some mixed combination of the 4 channels.

When you add more channels, the combinations grow exponentially and it becomes increasingly difficult to cover even a sensible subset of the possible combinations. On top of that, coding any one style of interleaver/de-interleaver is trivial, while coding one that can cover all combinations is far from trivial. This means that this feature will not be added any time soon.

**Q6 : What's the best format for storing temporary files?**

When you want to store temporary data there are a number of requirements;

* A simple, easy to parse header.
* The format must provide the fastest possible read and write rates (ie avoid conversions and encoding/decoding).
* The file format must be reasonably common and playable by most players.
* Able to store data in either endian-ness.

The format which best meets these requirements is AU, which allows data to be stored in any one of short, int, float and double (among others) formats.

For instance, if an application uses float data internally, its temporary files should use a format of (SF\_ENDIAN\_CPU | SF\_FORMAT\_AU | SF\_FORMAT\_FLOAT) which will store big endian float data in big endian CPUs and little endian float data on little endian CPUs. Reading and writing this format will not require any conversions or byte swapping regardless of the host CPU.

**Q7 : On Linux/Unix/MaxOS X, what's the best way of detecting the presence of libsndfile using autoconf?**

libsndfile uses the pkg-config (man pkg-config) method of registering itself with the host system. The best way of detecting its presence is using something like this in configure.ac (or configure.in):

PKG\_CHECK\_MODULES(SNDFILE, sndfile >= 1.0.2, ac\_cv\_sndfile=1, ac\_cv\_sndfile=0)  
  
 AC\_DEFINE\_UNQUOTED([HAVE\_SNDFILE],${ac\_cv\_sndfile},  
 [Set to 1 if you have libsndfile.])  
  
 AC\_SUBST(SNDFILE\_CFLAGS)  
 AC\_SUBST(SNDFILE\_LIBS)

This will automatically set the **SNDFILE\_CFLAGS** and **SNDFILE\_LIBS** variables which can be used in Makefile.am like this:

SNDFILE\_CFLAGS = @SNDFILE\_CFLAGS@  
 SNDFILE\_LIBS = @SNDFILE\_LIBS@

If you install libsndfile from source, you will probably need to set the **PKG\_CONFIG\_PATH** environment variable as suggested at the end of the libsndfile configure process. For instance on my system I get this:

-=-=-=-=-=-=-=-=-=-= Configuration Complete =-=-=-=-=-=-=-=-=-=-  
  
 Configuration summary :  
  
 Version : ..................... 1.0.5  
 Experimental code : ........... no  
  
 Tools :  
  
 Compiler is GCC : ............. yes  
 GCC major version : ........... 3  
  
 Installation directories :  
  
 Library directory : ........... /usr/local/lib  
 Program directory : ........... /usr/local/bin  
 Pkgconfig directory : ......... /usr/local/lib/pkgconfig  
  
 Compiling some other packages against libsndfile may require  
 the addition of "/usr/local/lib/pkgconfig" to the  
 PKG\_CONFIG\_PATH environment variable.

**Q8 : But I just want a simple Makefile! What do I do?**

The **pkg-config** program makes finding the correct compiler flag values and library location far easier. During the installation of libsndfile, a file named **sndfile.pc** is installed in the directory **${libdir}/pkgconfig** (ie if libsndfile is installed in **/usr/local/lib**, **sndfile.pc** will be installed in **/usr/local/lib/pkgconfig/**).

In order for pkg-config to find sndfile.pc it may be necessary to point the environment variable **PKG\_CONFIG\_PATH** in the right direction.

export PKG\_CONFIG\_PATH=/usr/local/lib/pkgconfig

Then, to compile a C file into an object file, the command would be:

gcc `pkg-config --cflags sndfile` -c somefile.c

and to link a number of objects into an executable that links against libsndfile, the command would be:

gcc `pkg-config --libs sndfile` obj1.o obj2.o -o program

**Q9 : How about adding the ability to write/read sound files to/from memory buffers?**

This has been added for version 1.0.13.

**Q10 : Reading a 16 bit PCM file as normalised floats and then writing them back changes some sample values. Why?**

This is caused by the fact that the conversion from 16 bit short to float is done by dividing by 32768 (0x8000 in hexadecimal) while the conversion from float to 16 bit short is done by multiplying by 32767 (0x7FFF in hex). So for instance, a value in a 16 bit PCM file of 20000 gets read as a floating point number of 0.6103515625 (20000.0 / 0x8000). Converting that back to a 16 bit short results in a value of 19999.3896484375 (0.6103515625 \* 0x7FFF) which then gets rounded down to 19999.

You will notice that for this particular case, the error is 1 in 20000 or 0.005%. Interestingly, for values of less than 16369, dividing by 0x8000 followed by multiplying by 0x7FFF and then rounding the result, gives back the original value. It turns out that as long as the host operating system supplies the 1999 ISO C Standard functions **lrintf** and **lrint** (or a replacement has been supplied) then the maximum possible error is 1 in 16369 or about 0.006%.

Regardless of the size of the error, the reason why this is done is rather subtle.

In a file containing 16 bit PCM samples, the values are restricted to the range [-32768, 32767] while we want floating point values in the range [-1.0, 1.0]. The only way to do this conversion is to do a floating point division by a value of 0x8000. Converting the other way, the only way to ensure that floating point values in the range [-1.0, 1.0] are within the valid range allowed by a 16 bit short is to multiply by 0x7FFF.

Some people would say that this is a severe short-coming of libsndfile. I would counter that anybody who is constantly converting back and forth between 16 bit shorts and normalised floats is going to suffer other losses in audio quality that they should also be concerned about.

Since this problem only occurs when converting between integer data on disk and normalized floats in the application, it can be avoided by using something other than normalized floats in the application. Alternatives to normalized floats are the **short** and **int** data types (ie using sf\_read\_short or sf\_read\_int) or using un-normalized floats (see  [SFC\_SET\_NORM\_FLOAT](http://www.mega-nerd.com/libsndfile/command.html#SFC_SET_NORM_FLOAT)).

Another way to deal with this problem is to consider 16 bit short data as a final destination format only, not as an intermediate storage format. All intermediate data (ie which is going to be processed further) should be stored in floating point format which is supported by all of the most common file formats. If floating point files are considered too large (2 times the size of a 16 bit PCM file), it would also be possible to use 24 bit PCM as an intermediate storage format (and which is also supported by most common file types).

**Q11 : I'm having problems with u-law encoded WAV files generated by libsndfile in Winamp. Why?**

This is actually a Winamp problem. The official Microsoft spec suggests that the 'fmt ' chunk should be 18 bytes. Unfortunately at least one of Microsoft's own applications (Sound Recorder on Win98 I believe) did not accept 18 bytes 'fmt ' chunks.

Michael Lee did some experimenting and found that:

I have checked that Windows Media Player 9, QuickTime Player 6.4,  
 RealOne Player 2.0 and GoldWave 5.06 can all play u-law files with  
 16-byte or 18-byte 'fmt ' chunk. Only Winamp (2.91) and foobar2000  
 are unable to play u-law files with 16-byte 'fmt ' chunk.

Even this is a very small sampling of all the players out there. For that reason it is probably not a good idea to change this now because there is the risk of breaking something that currently works.

**Q12 : I'm looking at sf\_read\*. What are items? What are frames?**

An itemtt> is a single sample of the data type you are reading; ie a single short value for sf\_read\_short or a single float for sf\_read\_float.

For a sound file with only one channel, a frame is the same as a item (ie a single sample) while for multi channel sound files, a single frame contains a single item for each channel.

Here are two simple, correct examples, both of which are assumed to be working on a stereo file, first using items:

#define CHANNELS 2  
 short data [CHANNELS \* 100] ;  
 sf\_count items\_read = sf\_read\_short (file, data, 200) ;  
 assert (items\_read == 200) ;

and now readng the exact same amount of data using frames:

#define CHANNELS 2  
 short data [CHANNELS \* 100] ;  
 sf\_count frames\_read = sf\_readf\_short (file, data, 100) ;  
 assert (frames\_read == 100) ;

**Q13 : Why can't libsndfile open this Sound Designer II (SD2) file?**

This is somewhat complicated. First some background.

SD2 files are native to the Apple Macintosh platform and use features of the Mac filesystem (file resource forks) to store the file's sample rate, number of channels, sample width and more. When you look at a file and its resource fork on Mac OS X it looks like this:

-rw-r--r-- 1 erikd erikd 46512 Oct 18 22:57 file.sd2  
 -rw-r--r-- 1 erikd erikd 538 Oct 18 22:57 file.sd2/rsrc

Notice how the file itself looks like a directory containing a single file named **rsrc**. When libsndfile is compiled for MacOS X, it should open (for write and read) SD2 file with resource forks like this without any problems. It will also handle files with the resource fork in a separate file as described below.

When SD2 files are moved to other platforms, the resource fork of the file can sometimes be dropped altogether. All that remains is the raw audio data and no information about the number of channels, sample rate or bit width which makes it a little difficult for libsndfile to open the file.

However, it is possible to safely move an SD2 file to a Linux or Windows machine. For instance, when an SD2 file is copied from inside MacOS X to a windows shared directory or a Samba share (ie Linux), MacOS X is clever enough to store the resource fork of the file in a separate hidden file in the same directory like this:

-rw-r--r-- 1 erikd erikd 538 Oct 18 22:57 .\_file.sd2  
 -rw-r--r-- 1 erikd erikd 46512 Oct 18 22:57 file.sd2

Regardless of what platform it is running on, when libsndfile is asked to open a file named **"foo"** and it can't recognize the file type from the data in the file, it will attempt to open the resource fork and if that fails, it then tries to open a file named **".\_foo"** to see if the file has a valid resource fork. This is the same regardless of whether the file is being opened for read or write.

In short, libsndfile should open SD2 files with a valid resource fork on all of the platforms that libsndfile supports. If a file has lost its resource fork, the only option is the open the file using the SF\_FORMAT\_RAW option and guessing its sample rate, channel count and bit width.

Occasionally, when SD2 files are moved to other systems, the file is [BinHexed](http://www.macdisk.com/binhexen.php3) which wraps the resource fork and the data fork together. For these files, it would be possible to write a BinHex parser but there is not a lot to gain considering how rare these BinHexed SD2 files are.

**Q14 : I'd like to statically link libsndfile to my closed source application. Can I buy a license so that this is possible?**

Unfortunately no. libsndfile contains code written by other people who have agreed that their code be used under the GNU LGPL but no more. Even if they were to agree, there would be significant difficulties in dividing up the payments fairly.

The **only** way you can legally use libsndfile as a statically linked library is if your application is released under the GNU GPL or LGPL.

**Q15 : My program is crashing during a call to a function in libsndfile. Is this a bug in libsndfile?**

libsndfile is being used by large numbers of people all over the world without any problems like this. That means that it is much more likely that your code has a bug than libsndfile. However, it is still possible that there is a bug in libsndfile.

To figure out whether it is your code or libsndfile you should do the following:

* Make sure you are compiling your code with warnings switched on and that you fix as many warnings as possible. With the GNU compiler (gcc) I would recommend at least **-W -Wall -Werror** which will force you to fix all warnings before you can run the code.
* Try using a memory debugger. [Valgrind](http://valgrind.kde.org/) on x86 Linux is excellent. [Purify](http://www.ibm.com/software/awdtools/purify/) also has a good reputation.
* If the code is clean after the above two steps and you still get a crash in libsndfile, then send me a small snippet of code (no more than 30-40 lines) which includes the call to sf\_open() and also shows how all variables passed to/returned from sf\_open() are defined.

**Q16 : Will you accept a fix for compiling libsndfile with compiler X?**

If compiler X is a C++ compiler then no. C and C++ are different enough to make writing code that compiles as valid C and valid C++ too difficult. I would rather spend my time fixing bugs and adding features.

If compiler X is a C compiler then I will do what I can as long as that does not hamper the correctness, portability and maintainability of the existing code. It should be noted however that libsndfile uses features specified by the 1999 ISO C Standard. This can make compiling libsndfile with some older compilers difficult.

**Q17 : Can libsndfile read/write files from/to UNIX pipes?**

Yes, libsndfile can read files from pipes. Unfortunately, the write case is much more complicated.

File formats like AIFF and WAV have information at the start of the file (the file header) which states the length of the file, the number of sample frames etc. This information must be filled in correctly when the file header is written, but this information is not reliably known until the file is closed. This means that libsndfile cannot write AIFF, WAV and many other file types to a pipe.

However, there is at least one file format (AU) which is specifically designed to be written to a pipe. Like AIFF and WAV, AU has a header with a sample frames field, but it is specifically allowable to set that frames field to 0x7FFFFFFF if the file length is not known when the header is written. The AU file format can also hold data in many of the standard formats (ie SF\_FORMAT\_PCM\_16, SF\_FORMAT\_PCM\_24, SF\_FORMAT\_FLOAT etc) as well as allowing data in both big and little endian format.

See also [FAQ Q6](#tyjcwt).

**Q18 : Is it possible to build a Universal Binary on Mac OS X?**

Yes, but you must do two separate configure/build/test runs; one on PowerPC and one on Intel. It is then possible to merge the binaries into a single universal binary using one of the programs in the Apple tool chain.

It is **not** possible to build a working universal binary via a single compile/build run on a single CPU.

The problem is that the libsndfile build process detects features of the CPU its being built for during the configure process and when building a universal binary, configure is only run once and that data is then used for both CPUs. That configure data will be wrong for one of those CPUs. You will still be able to compile libsndfile, and the test suite will pass on the machine you compiled it on. However, if you take the universal binary test suite programs compiled on one CPU and run them on the other, the test suite will fail.

Part of the problem is the the CPU endian-ness is detected at configure time. Yes, I know the Apple compiler defines one of the macros \_\_LITTLE\_ENDIAN\_\_ and \_\_BIG\_ENDIAN\_\_, but those macros are not part of the 1999 ISO C Standard and they are not portable.

Endian issues are not the only reason why the cross compiled binary will fail. The configure script also detects other CPU specific idiosyncrasies to provide more optimized code.

Finally, the real show stopper problem with universal binaries is the problem with the test suite. libsndfile contains a huge, comprehensive test suite. When you compile a universal binary and run the test suite, you only test the native compile. The cross compiled binary (the one with the much higher chance of having problems) cannot be tested.

Now, if you have read this far you're probably thinking there must be a way to fix this and there probably is. The problem is that its a hell of a lot of work and would require significant changes to the configure process, the internal code and the test suite. In addition, these changes must not break compilation on any of the platforms libsndfile is currently working on.

**Q19 : I have project files for Visual Studio / XCode / Whatever. Why don't you distribute them with libsndfile?**

There's a very good reason for this. I will only distribute things that I actually have an ability to test and maintain. Project files for a bunch of different compilers and Integrated Development Environments are simply too difficult to maintain.

The problem is that every time I add a new file to libsndfile or rename an existing file I would have to modify all the project files and then test that libsndfile still built with all the different compilers.

Maintaining these project files is also rather difficult if I don't have access to the required compiler/IDE. If I just edit the project files without testing them I will almost certainly get it wrong. If I release a version of libsndfile with broken project files, I'll get a bunch of emails from people complaining about it not building and have no way of fixing or even testing it.

I currently release sources that I personally test on Win32, Linux and MacOS X (PowerPC) using the compiler I trust (GNU GCC). Supporting one compiler on three (actually much more because GCC is available almost everywhere) platforms is doable without too much pain. I also release binaries for Win32 with instructions on how to use those binaries with Visual Studio. As a guy who is mainly interested in Linux, I'm not to keen to jump through a bunch of hoops to support compilers and operating systems I don't use.

So, I hear you want to volunteer to maintain the project files for Some Crappy Compiler 2007? Well sorry, that won't work either. I have had numerous people over the years offer to maintaining the project files for Microsoft's Visual Studio. Every single time that happened, they maintained it for a release or two and then disappeared off the face of the earth. Hence, I'm not willing to enter into an arrangement like that again.

**Q20 : Why doesn't libsndfile support MP3? Lots of other Open Source projects support it!**

MP3 is not supported for one very good reason; doing so requires the payment of licensing fees. As can be seen from  [mp3licensing.com](http://www.mp3licensing.com/royalty/software.html) the required royalty payments are not cheap.

Yes, I know other libraries ignore the licensing requirements, but their legal status is extremely dubious. At any time, the body selling the licenses could go after the authors of those libraries. Some of those authors may be students and hence wouldn't be worth pursuing.

However, libsndfile is released under the name of a company, Mega Nerd Pty Ltd; a company which has income from from libsamplerate licensing, libsndfile based consulting income and other unrelated consulting income. Adding MP3 support to libsndfile could place that income would be under legal threat.

Fortunately, Ogg Vorbis exists as an alternative to MP3. Support for Ogg Vorbis was added to libsndfile (mostly due to the efforts of John ffitch of the Csound project) in version 1.0.18.

**Q21 : How do I use libsndfile in a closed source or commercial program and comply with the license?**

Here is a checklist of things you need to do to make sure your use of libsndfile in a closed source or commercial project complies with the license libsndfile is released under, the GNU Lesser General Public License (LGPL):

* Make sure you are linking to libsndfile as a shared library (Linux and Unix systems), Dynamic Link Library (Microsoft Windows) or dynlib (Mac OS X). If you are using some other operating system that doesn't allow dynamically linked libraries, you will not be able to use libsndfile unless you release the source code to your program.
* In the licensing documentation for your program, add a statement that your software depends on libsndfile and that libsndfile is released under the GNU Lesser General Public License, either [version 2.1](http://www.gnu.org/licenses/lgpl-2.1.txt) or optionally [version 3](http://www.gnu.org/licenses/lgpl.txt).
* Include the text for both versions of the license, possibly as separate files named libsndfile\_lgpl\_v2\_1.txt and libsndfile\_lgpl\_v3.txt.

**Q22 : What versions of Windows does libsndfile work on?**

Currently the precompiled windows binaries are thoroughly tested on Windows XP. As such, they should also work on Win2k and Windows Vista. They may also work on earlier versions of Windows.

Since version 0.1.18 I have also been releasing precompiled binaries for Win64, the 64 bit version of Windows. These binaries have received much less testing than the 32 bit versions, but should work as expected. I'd be very interested in receiving feedback on these binaries.

**Q23 : I'm cross compiling libsndfile for another platform. How can I run the test suite?**

Since version 1.0.21 the top level Makefile has an extra make target, 'test-tarball'. Building this target creates a tarball called called:

libsndfile-testsuite-${host\_triplet}-${version}.tar.gz

in the top level directory. This tarball can then be copied to the target platform. Once untarred and test script test\_wrapper.sh can be run from the top level of the extracted tarball.

The libsndfile home page is here : <http://www.mega-nerd.com/libsndfile/>.

Version : 1.0.24