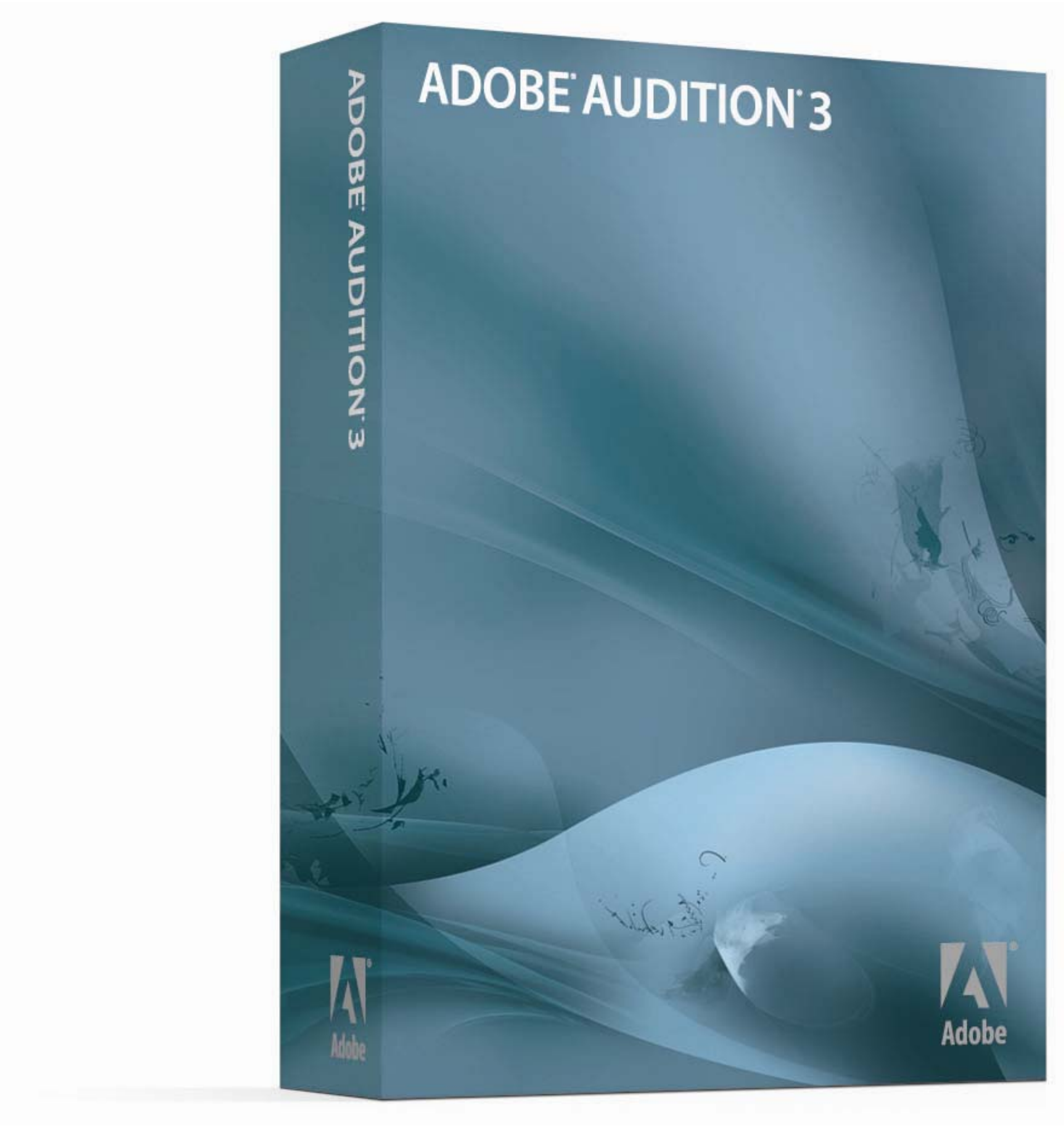

The File Format Filter (FLT) API
Version 3.0

The Adobe® Audition® Engineering team supplies the material in this document.

Notes on this release of Adobe® Audition® SDK

There have been relatively few content changes to the SDK sample code and documentation. References to “Cool” will persist, for the time being; after all, just because it’s an Adobe product doesn’t mean it’s not cool any more!

As we progress, we’ll rework the SDK, prioritizing our efforts based on developer requests. Please feel free to contact us with any API-related technical questions.

mailto:auditionsdk@adobe.com?subject=Audition_API_Question

Adobe® Audition® can support any number of file formats. New file formats can be written by following this API. A file filter is nothing more than a DLL with the following exported functions that Adobe® Audition® calls to do the reading (decoding) and writing (coding) of audio data.

When Adobe® Audition® starts, it checks the program directory for any files ending in .FLT. If it finds any, it checks to see if the file contains the function QueryCoolFilter, and if so, it calls that function. If the return value is C_VALIDLIBRARY, then it is assumed to be a valid file format DLL.

When reading an audio file, the function FilterUnderstandsFormat() is called first to see if your file filter can read the given file. If it can, then OpenFilterInput() is called, and then FilterGetFirstSpecialData() and FilterGetNextSpecialData() if the QF_READSPECIALFIRST flag is set. Audio data is then read in chunks of the size specified by calling ReadFilterInput(). If QF_READSPECIALLAST is set, then the special read functions are called after all audio data is read in. Finally, the CloseFilterInput() function is called when all is complete. The functions FilterOptions(), FilterOptionsString(), and FilterGetFileSize() may be called at any time while the input file is open.

When writing an audio file, the function FilterGetOptions() may be called at any time if the QF_HASOPTIONSBOX flag is set so the user can choose any filter specific options (e.g. A-Law, mu-Law or Linear in the case of .AU files, or 2-bit, 3-bit, or 4-bit compression in the case of DVI compressed files). The GetSuggestedSampleType() function is called to ensure that the file filter can write data in the format that Adobe® Audition® will be presenting. If a different sample type is specified, it will be converted to the desired type before being saved. OpenFilterOutput() is called to initiate the writing process. As in reading, the special write function FilterWriteSpecialData() may be called before or after writing the audio data depending on the flags set. Audio data is written by calling the WriteFilterOutput() function in blocks of the size specified by your filter at open time. Once everything is written, CloseFilterOutput() is called and writing is complete.

The FilterGetOptions() function should call up a dialog box for the user to choose the format options. The dialog box template should be a resource in the file filter DLL, thus it can be totally defined by the DLL author.

The sample AU wave file filter gives an example of writing a file filter that has multiple options. You can use these files as a template for creating your own file formats.

All functions should be declared as **__declspec(dllexport) FAR PASCAL**.

Overview of Functions called by Adobe® Audition®

QueryCoolFilter	Returns information about the type of file filter
FilterGetFileSize	Returns the number of audio data bytes that are allowed to read from the file.
FilterUnderstandsFormat	Returns TRUE if the given file is of a format that this filter understands
GetSuggestedSampleType	Returns the desired sample format for writing
FilterGetOptions	Brings up a dialog box to choose options for the filter if options supported
FilterOptions	Returns the options DWORD for the currently opened file
FilterOptionsString	Returns a readable text description of the filter options
FilterWriteSpecialData	Write a chunk of special non-audio data to the waveform file
FilterGetFirstSpecialData	Retrieve first item of special data from file
FilterGetNextSpecialData	Get the next chunk of special data from file
OpenFilterInput	Open the specified file for reading
ReadFilterInput	Read a specified number of bytes from the input audio stream
CloseFilterInput	Close the input file
OpenFilterOutput	Open the specified file for writing
WriteFilterOutput	Write a specified number of bytes to the output audio stream
CloseFilterOutput	Close the output file

Converting a legacy Cool Edit file filter for use with Adobe® Audition®

Older versions of Cool Edit (as far back as Cool Edit '95) supported the following file filter APIs: FilterCanReadSpecial(), FilterReadSpecial(), FilterCanWriteSpecial(), and FilterWriteSpecial(). These APIs are no longer supported by Adobe® Audition®.

These legacy APIs were replaced by the APIs FilterGetFirstSpecialData(), FilterGetNextSpecialData() and FilterWriteSpecialData(). This change was necessary in order to have Adobe® Audition® preserve special data entries that it had no knowledge of. Please see the descriptions for these new functions if you were reading or writing any extra non-audio info to the files.

Unicode support in Adobe® Audition® 2.0 and beyond

Starting with Adobe® Audition® 2.0, Unicode support was added to the text members of the COOLQUERY struct. The File Filter APIs which contain text pointers as function arguments have changed to Unicode. For example, the FilterUnderstandsFormat API changed from having a function argument of "LPSTR lpszFilename" to "LPWSTR lpwszFilename".

If you wish to build a file filter which **supports Unicode** and runs on Adobe® Audition® 2.0 and beyond, simply include Filters.h included within this SDK, and **return C_VALIDLIBRARY** (which is defined as 2000) in your implementation of QueryCoolFilter.

If you wish for your file filter to be **supported on older versions** of Adobe® Audition® (version 1.0 or 1.5) or any version of Cool Edit, then you will need to use Ascii function arguments in all of the file filter APIs mentioned in this document, and you must **return 1155** from QueryCoolFilter. By doing so, your file filter will still run properly with Adobe® Audition® 2.0 and 3.0 (and beyond), as well as any version of Cool Edit and older versions of Adobe® Audition®, however, you will not get Unicode support.

You cannot create a single file filter which supports both Unicode and non-Unicode interfaces. If you wish to take advantage of Unicode support for Adobe® Audition® 2.0 and beyond, but also want to support older versions, then you will need to create two different file filters, and install each to the corresponding program directories of the versions you are targeting.

int QueryCoolFilter(COOLQUERY * lpcq) Required

COOLQUERY * lpcq	*Structure to be filled with all information pertaining to the file filter.*

This function should fill the COOLQUERY structure with information about the file filter.

Returns: C_VALIDLABRARY if successful, zero otherwise. Note that if you wish for your file filter to support earlier versions of Adobe® Audition® or Cool Edit, you may want to return a different value. See comments in the previous section.

The COOLQUERY data structure is defined as:

WCHAR szName[24]	Textual description of file filter that will show in the File Open and File Save dialog boxes.
WCHAR szCopyright[80]	Any copyright information you care to put in for your DLL. This information is displayed as the file is being loaded or saved.
	Use the R_xxxx constants OR'ed together to form the sample rates supported by this filter.
WORD Quad32	(4-channel, 32-bit samples, not supported)
WORD Quad16	(4-Channel, 16-bit samples, not supported)
WORD Quad8	(4-Channel, 8-bit samples, not supported)
WORD Stereo8	Stereo, 8-bit samples
WORD Stereo12	(Stereo, 12-bit samples, not supported)
WORD Stereo16	Stereo, 16-bit samples
WORD Stereo24	(Stereo, 24-bit samples, not supported - specify as 32-bit instead)
WORD Stereo32	Stereo, 32-bit samples - formatted as float, scaled to range of +/-65536.0
WORD Mono8	Mono, 8-bit samples
WORD Mono12	(Mono, 12-bit samples, not supported)
WORD Mono16	Mono, 16-bit samples
WORD Mono24	(Mono, 24-bit samples, not supported - specify as 32-bit instead)
WORD Mono32	Mono, 32-bit samples - formatted as float, scaled to range of +/- 65536.0
DWORD dwFlags	Use the QF_xxxx flags ORed together to provide special information about what functions are supported by your filter, etc. and so on. See below for a list of QF_xxxx constants.
WCHAR szExt[4];	3-character default extension, in caps, followed by a NULL character
long lChunkSize;	Size of chuncks (in bytes) prefered when Adobe® Audition® calls your read and write functions
WCHAR szExt2[4];	Optional secondary 3-character default extension
WCHAR szExt3[4];	Optional extension
WCHAR szExt4[4];	And another optional extension, just in case.

Constants for Mono/Stereo/Quad 8/12/16/24/32 fields:

R_5500	Low quality, 5500 Hz
R_11025	AM radio quality, 11 KHz
R_22050	Radio quality, 22 KHz
R_32075	FM radio quality, 32 KHz (also valid for 32000 Hz)
R_44100	CD quality, 44.1 KHz
R_48000	DAT quality, 48 KHz

Constants for dwFlags field:

QF_RATEADJUSTABLE	File filter can handle adjustable rates, not just the standards
QF_CANSAVE	File filter supports saving (writing)
QF_CANLOAD	File loading (reading) supported
QF_UNDERSTANDSALL	File filter can read any file at all (used for PCM)
QF_READSPECIALFIRST	Special information should be read before data
QF_READSPECIALLAST	Special information should be read after data
QF_WRITESPECIALFIRST	Special information should be written before data
QF_WRITESPECIALLAST	Special information should be written after data
QF_HASOPTIONSBOX	This format supports multiple options
QF_NOASKFORCONVERT	Set to bypass asking for conversion if original in different rate, ie auto convert sample type
QF_NOHEADER	Set if this is a raw data format with no header
QF_CANDO32BITFLOATS	Set if file format can handle 32-bit floating point sample data for input
QF_CANOPENVIRTUAL	Set if data is in Intel 8-bit or 16-bit sample format, or floats and the GetDataOffset() function is implemented. Cool can then use files of this type directly without making a temporary file copy.

This function must return the value **C_VALIDLIBRARY** if everything is successful, otherwise it should return zero.

BOOL FilterUnderstandsFormat(LPWSTR lpwszFilename) Required

LPWSTR lpwszFilename	File name of file to test for understanding of.

Adobe® Audition® calls this function to determine if the given file can be read by the file filter. You may need to open the file and read in the first few bytes to verify that the header is correct for your format. For headerless formats, you may just need to check the filename extension to determine file validity.

Returns: TRUE if this file filter understands the format of lpwszFilename, FALSE otherwise.

void GetSuggestedSampleType(LONG * lplSamprate, WORD * lpwBitsPerSample, WORD * lpwChannels) Optional if Writing

LONG * lplSamprate	Sample rate
WORD * lpwBitsPerSample	Bits per Sample (8, 16, or 32)
WORD * lpwChannels	Channels (1 or 2)

Called with the current format of the waveform that is about to be saved. If this format is not supported, return the closest format that *is* supported, and Adobe® Audition® will prompt the user if they wish to convert to this format before saving. Return zero in any of the values to indicate that this parameter does not matter. For example, if this format handles all sample rates, set *lplSamprate to zero.

HANDLE OpenFilterOutput(LPWSTR lpwszFilename, LONG lSamprate, WORD wBitsPerSample, WORD wChannels, LONG lSize, LONG * lplChunkSize, DWORD dwOptions) Required if Writing

LPWSTR lpwszFilename	File to open for writing
LONG lSamprate	Sample rate in Hz
WORD wBitsPerSample	Sample size (currently 8 or 16)
WORD wChannels	Number of channels (currently 1 or 2)
LONG lSize	Total amount of data to be written, in bytes. *samples = lSize*wBitsPerSample/8/wChannels*
LONG * lplChunkSize	Adobe® Audition® will pass this value in *lBytes* to WriteFilterOutput()
DWORD dwOptions	Options string returned from FilterGetOptions(). Zero indicates the default should be used

A user defined type should be allocated and the handle returned. The user defined type should contain all variables that are used in the writing of files of this format. It will be passed in too all functions pertaining to writing. The output file should be opened for writing, overwriting any existing file of the same name since the user has already confirmed that the file name is correct. The user defined type should at the very least contain a handle to the opened file. The **lplChunkSize* parameter should be filled with the desired size of a chunk of audio data that WriteFilterOutput() would like. Set **lplChunkSize* to 16384 if the chunk size doesn't matter. The dwOptions parameter contains any specific options that the user chose when FilterGetOptions() was called. If the user never called this funciton, dwOptions will be zero, in which case some default should be used. This function will only be called if the QF_CANLOAD flag is set.

Returns: Valid handle to some internal output file structure, NULL file not opened for output for any reason.

void CloseFilterOutput(HANDLE hOutput) Required if Writing

HANDLE hOutput	Handle of user defined output structure

No more data is to be written out, and the file should be cleanly closed and the user defined structure referred to by hOutput should be freed. As far as Adobe® Audition® is concerned, the file has completed. The user may have hit Cancel during the write process, in which case this function may be called prematurely. If this is the case, the file should be cleaned up as much as possible so it can still be read, otherwise it should be deleted.

DWORD WriteFilterOutput(HANDLE hOutput, BYTE * lpbData, LONG lBytes) Required if Writing

HANDLE hOutput	Handle of user defined output structure
BYTE *lpbData	Data to be written (format was given in call to OpenFilterOutput())
LONG lBytes	Number of bytes in lpbData to be written

All data in lpbData should be written out to the file. The file handle should be part of the user defined structure specified by hOutput.

Returns: Number of bytes actually written. Once this value is less than lBytes, or is zero, then Adobe® Audition® will stop passing data to be written, and will call CloseFilterOutput().

HANDLE OpenFilterInput(LPWSTR lpwszFilename, LONG * lplSamprate, WORD * lpwBitsPerSample, WORD * lpwChannels, HWND hWnd, LONG * lplChunkSize)

Required if Reading

LPWSTR lpwszFilename	File to open for writing
LONG * lplSamprate	Sample rate in Hz
WORD * lpwBitsPerSample	Sample size (currently 8 or 16)
WORD * lpwChannels	Number of channels (currently 1 or 2)
HWND hWnd	Parent window - use to display error messages if needed
LONG * lplChunkSize	Adobe® Audition® will pass this value in *lBytes* to ReadFilterInput()

A user defined type should be allocated and the handle returned. The user defined type should contain all variables that are used in the reading of files of this format. It will be passed in too all functions pertaining to reading. The actual sample format should be filled into the lplSamprate, lpwBitsPerSample, and lpwChannels parameters. Fill lplChunkSize with the size of chunks you want ReadFilterInput to be called with. Use the hWnd value as a parent window if any custom dialogs or anything are used here. This function will only be called if the QF_CANSAVE flag is set.

Custom dialogs for configuring should be called from the FilterGetOptions() function instead of here. FilterGetOptions() will be called if a call to FilterOptionsString() or FilterOptions() returns zero. You should keep track of an options variable within your data structure, and return the value of that options variable in the FilterOptions() or FilterOptionsString call. If a value of zero is returned in that call, and QF_HASOPTIONS box is set, then FilterGetOptions() will be called for the user to choose the appropriate formatting options for this file.

If FilterGetOptions() is called, then it will be followed by a call to FilterSetOptions() so you can make the appropriate changes to your the internal data structure (eg making memory allocations based on the user's options that you would have done inside OpenFilterInput but couldn't because the actual data format was unknown at the time).

If a value of zero is returned for any of lplSamprate, lpwBitsPerSample, or lpwChannels, then the user will be queried as to what the proper value should be in an "Input Sample Rate" type dialog box automatically.

Returns: A handle to the user defined structure used in reading, or NULL if the file could not be opened.

DWORD FilterGetFileSize(HANDLE hInput)

Required if Reading

HANDLE hInput	Handle to user defined input structure, allocated during OpenFilterInput()

The size of the audio data in the file, in bytes, should be returned. This function is only called after OpenFilterInput succeeds.

Returns: The size of the audio data in bytes.

DWORD ReadFilterInput(HANDLE hInput, BYTE * lpbData, LONG lBytes)

Required if Reading

HANDLE hOutput	Handle of user defined input structure
BYTE *lpbData	Filled with newly read in data
LONG lBytes	Number of bytes in lpbData that should be read

lBytes bytes of audio data should be read into lpbData. The file handle should be part of the user defined structure specified by hInput.

Returns: Number of bytes actually read. Once this value is less than lBytes, or is zero, then Adobe® Audition® will stop passing data buffers to be read into, and will call CloseFilterInput().

void CloseFilterInput(HANDLE hInput)

Required if Reading

HANDLE hInput	Handle of user defined input structure

No more data is going to be read in. The file should be cleanly closed and the user defined structure referred to by hInput should be freed. As far as Adobe® Audition® is concerned, the file has completed reading in. The user may have hit Cancel during the read process, in which case this function may be called prematurely.

DWORD FilterGetOptions(HWND hWnd, HINSTANCE hInst, LONG lSamprate, WORD wChannels, WORD wBitsPerSample, DWORD dwDefaultOptions)

Optional

HWND hWnd	Handle to use as parent window of options dialog box
HINSTANCE hInst	Instance of your DLL so you can access an options dialog box template
LONG lSamprate	Sample rate of audio that will be saved
WORD wChannels	Number of channels in audio that will be saved
WORD wBitsPerSample	Bits per sample in audio that will be saved
DWORD dwDefaultOptions	Previous options settings, or zero to indicate defaults should be set.

If the QF_HASOPTIONSBOX flag is set, then the user may decide to change the file filter's parameters before saving. If so, this function is called. This function should in turn call up an options dialog box so the user may change the options. The options dialog box should default to the options specified by dwDefaultOptions. If dwDefaultOptions is zero, then the dialog box should choose appropriate defaults for all the settings. All options should fit within a single DWORD. The options DWORD should never be zero, since zero indicates that the default should be used.

FilterGetOptions() is called when the user presses the Options button in a File Save dialog, or during a file load if FilterOptions() or FilterOptionsString() returns zero.

Returns: Return zero if no options box is supported, otherwise return a valid DWORD containing all the options that the user chose. This options DWORD will be passed to OpenFilterOutput().

DWORD FilterOptions(HANDLE hInput)

Optional

HANDLE hInput	Handle to user defined input structure

If file filter supports multiple options, then this function should return the current options settings for the opened input file. This value will be used if the file is subsequently saved again in the same format, or if the Options dialog box is called up through FilterGetOptions().

Returns: Return zero if no options box is supported, othrwise return a valid DWORD containing the current options settings for the input file.

DWORD FilterOptionsString(HANDLE hInput, LPWSTR lpwszString)

Optional

HANDLE hInput	Handle to user defined input structure
LPWSTR lpwszString	String to hold text of options setting

The lpwszString parameter should be filled with a textual description of the current options settings. Currently this information is displayed for the user when "Show Info" is checked while choosing a file to open.

Return: Returns zero, and lpwszString set to the empty string, if no options box is supported. Otherwise, a valid DWORD containing the current options settings should be returned.

DWORD FilterSetOptions(HANDLE hInput, DWORD dwOptions, LONG lSamprate, WORD wChannels, WORD wBPS)

Optional

HANDLE hInput	Handle to user defined input structure
DWORD dwOptions	Options to set filter to
LONG lSamprate	Current sample rate
WORD wChannels	Current number of channels
WORD wBPS	Current bits per sample

If OpenFilterInput returns zero for the sample rate, channels, and bits per sample, but succeeds anyway, then the user is prompted for the sample format to use, then the FilterOptions function is called to allow the user to choose the options they wish to use to interpret the file (a good example is the PCM file format, where the data could be in any of a variety of formats). If FilterOptions succeeds with a non-zero return value (a valid option was chosen and cancel was not hit) then FilterSetOptions is called with the user's choices for options, sample rate, channels, and bits per sample to inform the file filter of all the options that should be used to interpret the file.

Return: This function should return the previous options setting.

DWORD FilterWriteSpecialData(HANDLE hOutput, LPCSTR lpszListType, LPCSTR lpszType, char * pData, DWORD dwSize)

Optional

HANDLE hOutput	Handle to user defined output type
LPCSTR lpszListType	Name of list in which this type lives (as in RIFF files)
LPCSTR lpszType	Name of this data
char * pData	Pointer to data
DWORD dwSize	Size of data

If the QF_WRITESPECIALFIRST or QF_WRITESPECIALLAST flag is set, this function is called to write special data to the wave file. Some file formats support extra information besides just waveform data. If this is the case, the requests from Adobe® Audition® to write the data can be asked before the audio data, or after, depending on the QF_ flag that is set.

Top level types (such as "cue ", "plst") should use "WAVE" as the lpszListType. Special text info strings (such as "ICMT", "ICOM", "ISFT", etc.) should use "INFO" as the lpszListType. The additional information types (such as "labl", "note", and "ltxt") should use "adtl" as the list type. The exact data format, and specifications of all the types and list types that can be used are identical to those found in the RIFF specification available from ftp.microsoft.com in the file riffnew.doc in the file mdrk.exe.

Return: 1 if all is OK, else 0 for an error.

DWORD FilterGetFirstSpecialData(HANDLE hInput, SPECIALDATA * psp)

Optional

HANDLE hInput	Handle to user defined input type
SPECIALDATA * psp	Pointer to SPECIALDATA structure which needs to be filled.

This API is used when any special information exists in the file and is handled by your file format. You can read in all the extra information during OpenFilterInput() and return the data as requested through this function. Adobe® Audition® will call this function either before or after the audio data is read depending on which of the QF_READSPECIALFIRST or QF_READSPECIALLAST flags are set.

SPECIALDATA Structure:

```
HANDLE hSpecialData;     // Handle to your own user defined data to keep track of state information or whatever
HANDLE hData;            // Actual handle to data that is being returned to the caller
DWORD  dwSize;           // size of data in handle
DWORD  dwExtra;          // optional extra data (usually a count)
char   szListType[8];    // Parent list type (usually "WAVE" or "INFO", or "adtl")
char   szType[8];        // Usually a four character code for data, but can be up to 7 chars
```

When this function is called, the first piece of extra information should be returned. The hSpecialData member is for your own use - to use as a count of the number of data items rerned to the caller so far, for example, so that when FilterGetNextSpecialData is called with this same structure, you can use the information in hSpecialData to determine which item of data should be returned next. All the members of this structure should be filled. All special data types basically mirror the equivalent RIFF chunks.

Return: 1 if the data item being returned is valid, 0 if this file has no extra information.

DWORD FilterGetNextSpecialData(HANDLE hInput, SPECIALDATA * psp)

Optional

HANDLE hInput	Handle to user defined input type
SPECIALDATA * psp	Pointer to SPECIALDATA structure which needs to be filled.

The SPECIALDATA structure should be filled out just as in FilterGetFirstSpecialData(). This function will continue to be called until a value of 0 is returned.

Return: 1 if this next data item was retrieved successfully, 0 to stop and say there are no more extra data items.

Structures

Following are the structures that are used in reading and writing files. Most have to do with the transfer of special types of extra data (data that is not audio). For more information on specific special types, see the RIFF specification, since the format of the data closely parallels that in the specification. Adobe® Audition® actually only supports 8 and 16 bit audio in mono and stereo formats - any sample rate. The COOLQUERY type allows for other formats though.

```
typedef DWORD FOURCC;                  // a four character code placed into a single DWORD (lsb first)

typedef struct riffspecialdata_t
{         HANDLE hSpecialData;         // Your own handle to use for any extra state info you may need to keep (such as a count
                                       // so you know how many special infos have been passed to the caller already.)
          HANDLE hData;                // Actual data handle
          DWORD  dwSize;               // size of data in handle
          DWORD  dwExtra;              // optional extra data (usually a count)
          char   szListType[8];        // Parent list type (usually "WAVE" or "INFO", or "adtl")
          char   szType[8];            // Usually a four character code for data, but can be up to 7 chars
} SPECIALDATA;

typedef struct coolquery_tag
{         WCHAR szName[24];            // Text description of file filter that shows in File Open and Save dialogs
          WCHAR szCopyright[80];       // Any copyright info, will display while file loads or saves
          WORD Quad32;                 // Bit field of supported rates of quadraphonic 32-bit audio
          WORD Quad16;                 // Bit field of supported rates of quadraphonic 16-bit audio
          WORD Quad8;                  // Bit field of supported rates of quadraphonic 8-bit audio
          WORD Stereo8;                // Bit field of supported rates of stereo 8-bit audio
          WORD Stereo12;               // Bit field of supported rates of stereo 12-bit audio
          WORD Stereo16;               // Bit field of supported rates of stereo 16-bit audio
          WORD Stereo24;               // Bit field of supported rates of stereo 24-bit audio
          WORD Stereo32;               // Bit field of supported rates of stereo 32-bit audio
          WORD Mono8;                  // Bit field of supported rates of mono 8-bit audio
          WORD Mono12;                 // Bit field of supported rates of mono 12-bit audio
          WORD Mono16;                 // Bit field of supported rates of mono 16-bit audio
          WORD Mono24;                 // Bit field of supported rates of mono 24-bit audio
          WORD Mono32;                 // Bit field of supported rates of mono 32-bit audio
          DWORD dwFlags;               // Various flags (see QueryCoolFIlter())
          WCHAR szExt[4];              // 3-letter filename extension for file format, zero terminated
          long lChunkSize;             // Desired data block size for reading or writing (in bytes)
          WCHAR szExt2[4];             // Alternate 3-character extensions to view when selecting this file type
          WCHAR szExt3[4];
          WCHAR szExt4[4];
}  COOLQUERY;
```

Definitions

```
#define R_5500      1               // Rate flags for supported sample types bitfields in COOLQUERY
#define R_11025     2
#define R_22050     4
#define R_32075     8
#define R_44100     16
#define R_48000     32

#define C_VALIDLIBRARY 1154         // Returned from QueryCoolFilter() to designate a valid file format

                                            // Flags for dwFlags field of COOLQUERY
#define QF_RATEADJUSTABLE           0x0001  // All sample rates are valid, not just the standard "R_" ones.
#define QF_CANSAVE                  0x0002  // Set if file filter can write files
#define QF_CANLOAD                  0x0004  // Set if file filter can read files
#define QF_UNDERSTANDSALL           0x0008  // Set if file filter reads any file format whatsoever (reserved for RAW PCM)
#define QF_READSPECIALFIRST         0x0010  // Set if extra info should be requested for before reading audio
#define QF_READSPECIALLAST          0x0020  // Set if extra info should be requested for after reading audio
#define QF_WRITESPECIALFIRST        0x0040  // Set if extra info should be written for before writing audio
#define QF_WRITESPECIALLAST         0x0080  // Set if extra info should be written for after writing audio
#define QF_HASOPTIONSBOX            0x0100  // Set if file format supports multiple options
#define QF_NOASKFORCONVERT          0x0200  // Set to convert sample type automatically, see GetSuggestedSampleType()
#define QF_NOHEADER                 0x0400  // Set if this file format does not have a header, if it is just raw data
#define QF_CANDO32BITFLOATS         0x0800  // set if file format can handle 32-bit sample data for input
#define QF_CANOPENVIRTUAL           0x1000  // Set if data is in Intel 8-bit or 16-bit sample format, or floats
                                            // and the GetDataOffset() function is implemented
```

Converting from 16-bit file filters

Following are the major differences between 16-bit and 32-bit file filters:

- The __export declarator should be changed to __declspec(dllexport) in all function definitions.
- LibMain and _WEP have been replaced by DllMain
- All occurrences of int should be changed to short
- Message procedures are now of the form MsgProc (HWND, UINT, WPARAM, LPARAM), and all functions using wParam and lParam should be verified.
- EXETYPE WINDOWS can be removed from the DEF file.
- The valid library return value is 1152.