MMedia for wxWindows

Guilhem Lavaux

March 2000

Contents

Introduction

The MMedia wxWindows extension is a wxWindows library which provides you a full set of multimedia classes including sound recording/playing, cd audio playing and video playing. The API is portable and can be used on any supported systems with the insurance the behaviour will be the same.

File structure

These are the files that comprise the mmedia library.

sndbase.h Header for wxSoundStream base class and wxSoundFormat base class.

sndbase.cpp Basic objects implementation.

sndfile.h wxSoundFileStream base class header.

sndfile.cpp wxSoundFileStream base class implementation.

sndpcm.h wxSoundFormatPcm class header.

sndpcm.cpp wxSoundFormatPcm class implementation.

sndcpcm.h wxSoundCodecPcm class header (PCM converter).

sndcpcm.cpp wxSoundCodecPcm class implementation (PCM converter).

sndulaw.h

sndulaw.cpp

sndg72x.h

sndg72x.cpp

sndoss.h

sndoss.cpp

sndesd.h

sndesd.cpp

sndwin.h

sndwin.cpp

cdbase.h

cdbase.cpp

cdunix.h

cdunix.cpp

cdwin.h

cdwin.cpp

vidbase.h

vidbase.cpp

vidxanm.h

vidxanm.cpp

vidwin.h

vidwin.cpp

MMboard: a sample MMedia application

To be written.

Class reference

These are the main Mmedia classes.

wxCDAudio

Derived from

wxObject

Data structures

```
typedef struct wxCDtime {
  wxUint8 track
};


typedef enum  PLAYING, PAUSED, STOPPED  CDstatus
```

wxCDAudioLinux

Derived from

wxCDAudio (p. 4)

Data structures

wxCDAudioLinux::wxCDAudioLinux

wxCDAudioLinux()

wxCDAudioLinux(**const char*** *dev_name*)

wxCDAudioLinux::~wxCDAudioLinux

~wxCDAudioLinux()

wxCDAudioLinux::Play

bool Play(**const wxCDtime&** *beg_time*, **const wxCDtime&** *end_time*)

wxCDAudioLinux::Pause

bool Pause()

wxCDAudioLinux::Resume

bool Resume()

wxCDAudioLinux::GetStatus

CDstatus GetStatus()

wxCDAudioLinux::GetTime

wxCDtime GetTime()

wxCDAudioLinux::GetToc

CDtoc& GetToc()

wxCDAudioLinux::Ok

bool Ok() **const**

wxCDAudioLinux::OpenDevice

void OpenDevice(**const char*** *dev_name*)

wxCDAudioWin

Derived from

wxCDAudio (p. 4)

Data structures

```
typedef struct CDAW\_Internal {
  MCIDEVICEID dev\_id
};
```

wxCDAudioWin::wxCDAudioWin

wxCDAudioWin()

wxCDAudioWin(**const char*** *dev_name*)

wxCDAudioWin::~wxCDAudioWin

~wxCDAudioWin()

wxCDAudioWin::Play

bool Play(**const wxCDtime&** *beg_time*, **const wxCDtime&** *end_time*)

wxCDAudioWin::Pause

bool Pause()

wxCDAudioWin::Resume

bool Resume()

wxCDAudioWin::GetStatus

CDstatus GetStatus()

wxCDAudioWin::GetTime

wxCDtime GetTime()

wxCDAudioWin::GetToc

const CDtoc& GetToc()

wxCDAudioWin::Ok

bool Ok() **const**

wxCDAudioWin::PrepareToc

void PrepareToc()

CDtoc

Table of contents manager

Derived from

No base class

Data structures

CDtoc::CDtoc

CDtoc(**wxCDtime&** *tot_tm*, **wxCDtime*** *trks_tm*, **wxCDtime*** *trks_pos*)

CDtoc::GetTrackTime

wxCDtime GetTrackTime(**wxUint8** *track*) **const**

Returns the length of the specified track track: track to get length

CDtoc::GetTrackPos

wxCDtime GetTrackPos(**wxUint8** *track*) **const**

Returns the position of the specified track track: track to get position

CDtoc::GetTotalTime

wxCDtime GetTotalTime() **const**

Returns the total time

wxCDAudio::wxCDAudio

wxCDAudio()

wxCDAudio::~wxCDAudio

~wxCDAudio()

wxCDAudio::Play

bool Play(**const wxCDtime&** *beg_play*, **const wxCDtime&** *end_play*)

Play audio at the specified position

bool Play(**const wxCDtime&** *beg_play*)

Play audio from the specified to the end of the CD audio

bool Play(**wxUint8** *beg_track*, **wxUint8** *end_track = 0*)

wxCDAudio::Pause

bool Pause()

Pause the audio playing

wxCDAudio::Resume

bool Resume()

Resume a paused audio playing

wxCDAudio::GetStatus

CDstatus GetStatus()

Get the current CD status

wxCDAudio::GetTime

wxCDtime GetTime()

Get the current playing time

wxCDAudio::GetToc

const CDtoc& GetToc()

Returns the table of contents

wxCDAudio::Ok

bool Ok() **const**

CD ok

wxSoundStream

Base class for sound streams

Derived from

No base class

Include files

<wx/mmedia/sndbase.h>

Data structures

wxSoundStream errors

wxSOUND_NOERR	No error occurred
wxSOUND_IOERR	An input/output error occurred, it may concern either a driver or a file
wxSOUND_INVFRMT	The sound format passed to the function is invalid. Generally, it means that you passed out of range values to the codec stream or you don't pass the right sound format object to the right sound codec stream.
wxSOUND_INVDEV	Invalid device. Generally, it means that the

	sound stream didn't manage to open the device driver due to an invalid parameter or to the fact that sound is not supported on this computer.
wxSOUND_NOEXACT	No exact matching sound codec has been found for this sound format. It means that the sound driver didn't manage to setup the sound card with the specified values.
wxSOUND_NOCODEC	No matching codec has been found. Generally, it may happen when you call wxSoundRouterStream::SetSoundFormat().
wxSOUND_MEMERR	Not enough memory.

C callback for wxSound event

When a sound event is generated, it may either call the internal sound event processor (which can be inherited) or call a C function. Its definition is:

```
typedef void (*wxSoundCallback)(wxSoundStream *stream, int evt,
                                void *cdata);
```

The **stream** parameter represents the current wxSoundStream.

The **evt** parameter represents the sound event which is the cause of the calling. (See *wxSound events* (p. 8)).

The **cdata** parameter represents the user callback data which were specified when the user called *wxSoundStream::Register* (p. 12).

Note: There are two other ways to catch sound events: you can inherit the sound stream and redefine *wxSoundStream::OnSoundEvent* (p. 13), or you can reroute the events to another sound stream using *wxSoundStream::SetEventHandler* (p. 12).

wxSound streaming mode

The wxSoundStream object can work in three different modes. These modes are specified at the call to *wxSoundStream::StartProduction* (p. 12) and cannot be changed until you call *wxSoundStream::StopProduction* (p. 12).

The **wxSOUND_INPUT** mode is the recording mode. It generates **wxSOUND_INPUT** events and you cannot use wxSoundStream::Write().

The **wxSOUND_OUTPUT** mode is the playing mode. It generates **wxSOUND_OUTPUT** events and you cannot use wxSoundStream::Read().

The **wxSOUND_DUPLEX** mode activates the full duplex mode. The full duplex requires you to make synchronous call to *wxSoundStream::Read* (p. 10) and *wxSoundStream::Write* (p. 10). This means that you must be careful with realtime problems. Each time you call Read you must call Write.

wxSoundStream events

The sound events are generated when the sound driver (or the sound stream) completes a previous sound buffer. There are two possible sound events and two meanings.

The **wxSOUND_INPUT** event is generated when the sound stream has a new input buffer ready to be read. You know that you can read a buffer of the size*GetBestSize()* (p. 11) without blocking.

The **wxSOUND_OUTPUT** event is generated when the sound stream has completed a previous buffer. This buffer has been sent to the sound driver and it is ready to process a new buffer. Consequently, *Write* (p. 10) will not block too.

wxSoundStream::wxSoundStream

wxSoundStream()

Default constructor.

wxSoundStream::~wxSoundStream

~wxSoundStream()

Destructor. The destructor stops automatically all started production and destroys any temporary buffer.

wxSoundStream::Read

wxSoundStream& Read(**void*** *buffer*, **wxUint32** *len*)

Reads *len* bytes from the sound stream. This call may block the user so use it carefully when you need to intensively refresh the GUI. You may be interested by sound events: see *wxSoundStream::OnSoundEvent* (p. 13).

It is better to use the size returned by *wxSoundStream::GetBestSize* (p. 11): this may improve performance or accuracy of the sound event system.

Parameters

len

> *len* is expressed in bytes. If you need to do conversions between bytes and seconds use wxSoundFormat. See *wxSoundFormatBase* (p. 17), *wxSoundStream::GetSoundFormat* (p. 11).

data

> Data in *buffer* are coded using the sound format attached to this sound stream. The format is specified with *SetSoundFormat* (p. 11).

wxSoundStream::Write

wxSoundStream& Write(const void* *buffer*, **wxUint32** *len*)

Writes *len* bytes to the sound stream. This call may block the user so use it carefully. You may be interested by sound events: see*wxSoundStream::OnSoundEvent* (p. 13).

It is better to use the size returned by *wxSoundStream::GetBestSize* (p. 11): this may improve performance or accuracy of the sound event system.

Parameters

len

This is expressed in bytes. If you need to do conversions between bytes and seconds use wxSoundFormat. See *wxSoundFormatBase* (p. 17), *wxSoundStream::GetSoundFormat* (p. 11).

buffer

Data in *buffer* are coded using the sound format attached to this sound stream. The format is specified with *SetSoundFormat* (p. 11).

wxSoundStream::GetBestSize

wxUint32 GetBestSize() const

This function returns the best size for IO calls. The best size provides you a good alignment for data to be written (or read) to (or from) the sound stream. So, when, for example, a sound event is sent, you are sure the sound stream will not block for this buffer size.

wxSoundStream::SetSoundFormat

bool SetSoundFormat(const wxSoundFormatBase& *format*)

SetSoundFormat is one of the key function of the wxSoundStream object. It specifies the sound format the user needs. SetSoundFormat tries to apply the format to the current sound stream (it can be a sound file or a sound driver). Then, either it manages to apply it and it returns **TRUE**, or it could not and it returns **FALSE**. In this case, you must check the error with *wxSoundStream::GetError* (p. 12). See *wxSoundStream errors section* (p. 8) for more details.

Note

The **format** object can be destroyed after the call. The object does not need it.

Note

If the error is **wxSOUND_NOTEXACT**, the stream tries to find the best approaching format and setups it. You can check the format which it applied with *wxSoundStream::GetSoundFormat* (p. 11).

wxSoundStream::GetSoundFormat

wxSoundFormatBase& GetSoundFormat() const

It returns a reference to the current sound format of the stream represented by a wxSoundFormatBase object. This object *must not* be destroyed by anyone except the stream itself.

wxSoundStream::SetCallback

void Register(int *evt*, **wxSoundCallback** *cbk*, **void*** *cdata*)

It installs a C callback for wxSoundStream events. The C callbacks are still useful to avoid hard inheritance. You can install only one callback per event. Each callback has its callback data.

wxSoundStream::StartProduction

bool StartProduction(int *evt*)

StartProduction starts the sound streaming. *evt* may be one of **wxSOUND_INPUT**, **wxSOUND_OUTPUT** or **wxSOUND_DUPLEX**. You cannot specify several flags at the same time. Starting the production may automaticaly in position of buffer underrun (only in the case you activated recording). Actually this may happen the sound IO queue is too short. It is also advised that you fill quickly enough the sound IO queue when the driver requests it (through a wxSoundEvent).

wxSoundStream::StopProduction

bool StopProduction()

I stops the async notifier and the sound streaming straightly.

wxSoundStream::SetEventHandler

void SetEventHandler(wxSoundStream* *handler*)

Sets the event handler: if it is non-null, all events are routed to it.

wxSoundStream::GetError

wxSoundError GetError() const

It returns the last error which occurred.

wxSoundStream::GetLastAccess

wxUint32 GetLastAccess() const

It returns the number of bytes which were effectively written to/read from the sound stream.

wxSoundStream::QueueFilled

bool QueueFilled() **const**

It returns whether the sound IO queue is full. When it is full, the next IO call will block until the IO queue has at least one empty entry.

wxSoundStream::OnSoundEvent

void OnSoundEvent(**int** *evt*)

It is called by the wxSoundStream when a new sound event occurred.

wxSoundFileStream

Base class for file coders/decoders. This class is not constructor (it is an abstract class).

Derived from

wxSoundStream (p. 8)

Include file

wx/sndfile.h

Data structures

wxSoundFileStream::wxSoundFileStream

wxSoundFileStream(**wxInputStream&** *stream*, **wxSoundStream&** *io_sound*)

It constructs a new file decoder object which will send audio data to the specified sound stream. The *stream* is the input stream to be decoded. The*io_sound* is the destination sound stream. Once it has been constructed, you cannot change any of the specified streams nor the direction of the stream.

You will have access to the playback functions.

wxSoundFileStream(**wxOutputStream&** *stream*, **wxSoundStream&** *io_sound*)

It constructs a new file coder object which will get data to be recorded from the specified sound stream. The *stream* is the output wxStream. The *io_sound*is the source sound stream of the audio data. Once it has been constructed, you cannot change any of the specified streams nor the direction of the stream.

wxSoundFileStream::~wxSoundFileStream

~wxSoundFileStream()

It destroys the current sound file codec.

wxSoundFileStream::Play

bool Play()

It starts playing the file. The playing begins, in background in nearly all cases, after the return of the function. The codec returns to a **stopped** state when it reaches the end of the file. On success, it returns TRUE.

wxSoundFileStream::Record

bool Record(**wxUint32** *time*)

It starts recording data from the sound stream and writing them to the output stream. You have to precise the recording length in parameter. This length is expressed in seconds. If you want to control the record length (using *Stop* (p. 14)), you can set it to wxSOUND_INFINITE_TIME.

On success, it returns TRUE.

wxSoundFileStream::Stop

bool Stop()

It stops either recording or playing. Whatever happens (even unexpected errors), the stream is stopped when the function returns. When you are in recording mode, the file headers are updated and flushed if possible (ie: if the output stream is seekable).

On success, it returns TRUE.

wxSoundFileStream::Pause

bool Pause()

The file codec tries to pause the stream: it means that it stops audio production but keep the file pointer at the place.

If the file codec is already paused, it returns FALSE.

On success, it returns TREE.

wxSoundFileStream::Resume

bool Resume()

When the file codec has been paused using*Pause* (p. 14), you could be interrested in resuming it. This is the goal of this function.

wxSoundFileStream::IsStopped

bool IsStopped() **const**

It returns TRUE when the stream is stopped, in another case it returns FALSE.

wxSoundFileStream::IsPaused

bool IsPaused() **const**

It returns TRUE when the stream is paused, in another case it returns FALSE.

wxSoundFileStream::StartProduction

bool StartProduction(**int** *evt*)

It is really not advised you call this function. From the wxSoundFileStream point of view it is an internal function. Internally, it is called after the file stream has been prepared to be played or to receive audio data and when it wants to start processing audio data.

wxSoundFileStream::StopProduction

bool StopProduction()

As for *StartProduction* (p. 15), it is not advised for you to call this function. It is called by*Stop* (p. 14) when it needs to stop the audio data processing.

wxSoundFileStream::GetLength

wxUint32 GetLength()

It returns the audio data length of the file stream. This length is expressed in bytes. If you need the length in seconds, you will need to use*GetSoundFormat* (p. 11) and*GetTimeFromBytes* (p. 18).

wxSoundFileStream::GetPosition

wxUint32 GetPosition()

It returns the current position in the soundfile stream. The position is expressed in bytes. If you need the length in seconds, you will need to use*GetSoundFormat* (p. 11) and*GetTimeFromBytes* (p. 18).

wxSoundFileStream::SetPosition

wxUint32 SetPosition(**wxUint32** *new_position*)

It sets the current in the soundfile stream. The position *new_position* must be expressed in bytes. You can get a length/position in bytes from a time value using*GetSoundFormat* (p. 11) and *GetTimeFromBytes* (p. 18).

On success, it returns TRUE.

Warning

Some wxStream may not be capable to support this function as it may not support the seekable functionnality. If this happens, it returns FALSE and leave the stream at the same position.

wxSoundFileStream::Read

wxSoundStream& Read(**void*** *buffer*, **wxUint32** *len*)

You can obtain the audio data encoded in the file using this function. But it must be considered as an internal function. Used carelessly, it may corrupt the current state of the stream. Data are returned using in the original file coding (You must use a sound format object to decode it).

wxSoundFileStream::Write

wxSoundStream& Write(**const void*** *buffer*, **wxUint32** *len*)

You can put encoded audio data to the file using this function. But it must be considered as an internal function. Used carelessly, it may corrupt the current state of the stream. Data must be coded with the specified file coding (You must use a sound format object to do this).

wxSoundFileStream::SetSoundFormat

bool SetSoundFormat(**const wxSoundFormatBase&** *format*)

wxSoundFileStream::GetCodecName

wxString GetCodecName() **const**

This function returns the Codec name. This is useful for those who want to build a player (But also in some other case).

wxSoundFileStream::CanRead

bool CanRead()

You should use this function to test whether this file codec can read the stream you passed to it.

wxSoundFileStream::PrepareToPlay

bool PrepareToPlay()

It is called by wxSoundFileStream to prepare the specific file loader to prepare itself to play the file. Actually, this includes reading headers and setting the various parameters of the sound format. This should not be called by an external user but it should be implemented when you inherit wxSoundFileStream to build a new codec.

It must return when the file is identified and the parameters have been set. In all other

cases, you must return FALSE.

wxSoundFileStream::PrepareToRecord

bool PrepareToRecord(**wxUint32** *time*)

wxSoundFileStream::FinishRecording

bool FinishRecording()

wxSoundFileStream::RepositionStream

bool RepositionStream(**wxUint32** *position*)

This is called by wxSoundFileStream::SetPosition to seek the input stream to the right position. This must be overidden by the file codec class. The position is relative to the beginning of the samples. If it is impossible (as for a piped input stream), you must return FALSE.

wxSoundFileStream::FinishPreparation

void FinishPreparation(**wxUint32** *len*)

This is an internal function but it must called by the file codec class when the "playing" preparation is finished and you know the size of the stream. If it is an *infinite* stream, you should set this to wxSOUND_INFINITE_TIME.

wxSoundFileStream::GetData

wxUint32 GetData(**void*** *buffer*, **wxUint32** *len*)

This is called by wxSoundFileStream when it needs to get new sound data to send to the device driver (or to a conversion codec). This must be eventually overidden by the file codec class. The default behaviour is simply to read from the input stream.

wxSoundFileStream::PutData

wxUint32 PutData(**const void*** *buffer*, **wxUint32** *len*)

This is called by wxSoundFileStream when it needs to put new sound data received from the device driver (or from a conversion codec). This must be eventually overidden by the file codec class. The default behaviour is simply to write to the input stream.wxSoundFormatBase

Base class for sound format specification

Derived from

No base class

Data structures

```
typedef enum
  wxSOUND_NOFORMAT,
  wxSOUND_PCM,
  wxSOUND_ULAW,
  wxSOUND_G72X,
  wxSOUND_MSADPCM
 wxSoundFormatType
```

wxSoundFormatType: it specifies the format family of the sound data which will be passed to the stream.

wxSoundFormatBase::wxSoundFormatBase

wxSoundFormatBase()

wxSoundFormatBase::~wxSoundFormatBase

~wxSoundFormatBase()

wxSoundFormatBase::GetType

wxSoundFormatType GetType() **const**

It returns a "standard" format type.

wxSoundFormatBase::Clone

wxSoundFormatBase* Clone() **const**

It clones the current format.

wxSoundFormatBase::GetTimeFromBytes

wxUint32 GetTimeFromBytes(**wxUint32** *bytes*) **const**

wxSoundFormatBase::GetBytesFromTime

wxUint32 GetBytesFromTime(**wxUint32** *time*) **const**

wxSoundFormatBase::operator!=

bool operator!=(**const wxSoundFormatBase&** *frmt2*) **const**

Topic overviews

The following sections describe particular topics.

MMedia extension overview

To be written.

Bugs

These are the known bugs.

- No bugs

Change log

Index