

#\$+K!

MMedia for wxWindows

Guilhem Lavaux

March 2000

Contents

[Introduction](#)

[MMboard: a sample MMedia application](#)

[Class reference](#)

[Topic overviews](#)

[Bugs](#)

[Change log](#)

^Contents

^Contents

^browse00001

^K Contents

^DisableButton("Up")

\$#+K! 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

lntroduction

topic0

browse00002

K Introduction

DisableButton("Up")

MMboard: a sample MMedia application

To be written.

MMboard: a sample MMedia application
mboard
rowse00004
K MMboard a sample MMedia application
isableButton("Up")

^{\$#+K!}**Class reference**

These are the main Mmedia classes.

[wxCDAudio](#)

[wxCDAudioLinux](#)

[wxCDAudioWin](#)

[CDtoc](#)

[wxSoundStream](#)

[wxSoundFileStream](#)

[wxSoundFormatBase](#)

^Class reference

^Classref

^browse00005

^K Class reference

^DisableButton("Up")

Topic overviews

The following sections describe particular topics.

[MMedia extension overview](#)

Topic overviews

overviews

rowse00094

Topic overviews

isableButton("Up")

Bugs

These are the known bugs.

{bmc bullet.bmp} No bugs

Bugs
bugs
rowse00096
K Bugs
isableButton("Up")

Change log

Change log
topic2
rowse00097
Change log
isableButton("Up")

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

File structure

topic1

browse00003

^K File structure

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('mmedia.hlp', `topic0')")

cdwin.h

cdwin.cpp

vidbase.h

vidbase.cpp

vidxanm.h

vidxanm.cpp

vidwin.h

vidwin.cpp

`wxCDAudio`

Derived from

`wxObject`

Data structures

```
typedef struct wxCDtime {
    wxUint8 track
};

typedef enum    PLAYING, PAUSED, STOPPED    CDstatus
```

```
wxCDAudio
wxcdaudio
browse00006
K wxCDAudio
E enableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `classref')")
```

\$#+K! **wxCDAudioLinux**

Derived from

wxCDAudio

Data structures

Members

wxCDAudioLinux::wxCDAudioLinux
wxCDAudioLinux::~wxCDAudioLinux
wxCDAudioLinux::Play
wxCDAudioLinux::Pause
wxCDAudioLinux::Resume
wxCDAudioLinux::GetStatus
wxCDAudioLinux::GetTime
wxCDAudioLinux::GetToc
wxCDAudioLinux::Ok
wxCDAudioLinux::OpenDevice

^wxCDAudioLinux

^wxcdaudiolinux

^browse00007

^K wxCDAudioLinux

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `classref')")

\$#+K! **wxCDAudioWin**

Derived from

wxCDAudio

Data structures

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

Members

wxCDAudioWin::wxCDAudioWin
wxCDAudioWin::~~wxCDAudioWin
wxCDAudioWin::Play
wxCDAudioWin::Pause
wxCDAudioWin::Resume
wxCDAudioWin::GetStatus
wxCDAudioWin::GetTime
wxCDAudioWin::GetToc
wxCDAudioWin::Ok
wxCDAudioWin::PrepareToc

^wxCDAudioWin

^wxcdaudiowin

^browse00018

^K wxCDAudioWin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `classref')")

\$#+K! **CDtoc**

Table of contents manager

Derived from

No base class

Data structures

Members

[CDtoc::CDtoc](#)
[CDtoc::GetTrackTime](#)
[CDtoc::GetTrackPos](#)
[CDtoc::GetTotalTime](#)
[wxCDAudio::wxCDAudio](#)
[wxCDAudio::~~wxCDAudio](#)
[wxCDAudio::Play](#)
[wxCDAudio::Pause](#)
[wxCDAudio::Resume](#)
[wxCDAudio::GetStatus](#)
[wxCDAudio::GetTime](#)
[wxCDAudio::GetToc](#)
[wxCDAudio::Ok](#)

^cDtoc

^cdtoc

^browse00029

^K CDtoc

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `classref')")

`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

`wxSoundStream`

`wxsoundstream`

`rowse00043`

`wxSoundStream`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('mmedia.hlp', `classref')")`

(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](#)).

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

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

wxSound streaming mode

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

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](#) and [wxSoundStream::Write](#). 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\(\)](#) 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](#) will not block too.

Members

[wxSoundStream::wxSoundStream](#)
[wxSoundStream::~~wxSoundStream](#)
[wxSoundStream::Read](#)
[wxSoundStream::Write](#)
[wxSoundStream::GetBestSize](#)

wxSoundStream::SetSoundFormat
wxSoundStream::GetSoundFormat
wxSoundStream::SetCallback
wxSoundStream::StartProduction
wxSoundStream::StopProduction
wxSoundStream::SetEventHandler
wxSoundStream::GetError
wxSoundStream::GetLastAccess
wxSoundStream::QueueFilled
wxSoundStream::OnSoundEvent

\$#+K! **wxSoundFileStream**

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

Derived from

[wxSoundStream](#)

Include file

wx/sndfile.h

Data structures

Members

[wxSoundFileStream::wxSoundFileStream](#)
[wxSoundFileStream::~~wxSoundFileStream](#)
[wxSoundFileStream::Play](#)
[wxSoundFileStream::Record](#)
[wxSoundFileStream::Stop](#)
[wxSoundFileStream::Pause](#)
[wxSoundFileStream::Resume](#)
[wxSoundFileStream::IsStopped](#)
[wxSoundFileStream::IsPaused](#)
[wxSoundFileStream::StartProduction](#)
[wxSoundFileStream::StopProduction](#)
[wxSoundFileStream::GetLength](#)
[wxSoundFileStream::GetPosition](#)
[wxSoundFileStream::SetPosition](#)
[wxSoundFileStream::Read](#)
[wxSoundFileStream::Write](#)
[wxSoundFileStream::SetSoundFormat](#)
[wxSoundFileStream::GetCodecName](#)
[wxSoundFileStream::CanRead](#)
[wxSoundFileStream::PrepareToPlay](#)
[wxSoundFileStream::PrepareToRecord](#)
[wxSoundFileStream::FinishRecording](#)
[wxSoundFileStream::RepositionStream](#)
[wxSoundFileStream::FinishPreparation](#)
[wxSoundFileStream::GetData](#)
[wxSoundFileStream::PutData](#)

^wxSoundFileStream

^wxsoundfilestream

^browse00059

^K wxSoundFileStream

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('mmedia.hlp', `classref')")

\$#+K! **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.

Members

[wxSoundFormatBase::wxSoundFormatBase](#)
[wxSoundFormatBase::~~wxSoundFormatBase](#)
[wxSoundFormatBase::GetType](#)
[wxSoundFormatBase::Clone](#)
[wxSoundFormatBase::GetTimeFromBytes](#)
[wxSoundFormatBase::GetBytesFromTime](#)
[wxSoundFormatBase::operator!=](#)

^wxSoundFormatBase

^wxsoundformatbase

^browse00086

^K wxSoundFormatBase

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `classref')")

MMedia extension overview

To be written.

MMedia extension overview

medoverview

rowse00095

MMedia extension overview

nableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp`, `overviews`)"

`$#+K! wxCDAudioLinux::wxCDAudioLinux`

`wxCDAudioLinux()`^K

`wxCDAudioLinux(const char* dev_name)`^K

`wxCDAudioLinux::wxCDAudioLinux`

`wxcdaudiolinuxwxcdaudiolinux`

`browse00008`

`K wxCDAudioLinux wxCDAudioLinux`

**`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(mmedia.hlp',
`wxcdaudiolinux')")`**

`K wxCDAudioLinux`

`K wxCDAudioLinux`

\$#+K!wxCDAudioLinux::~~wxCDAudioLinux

~wxCDAudioLinux()^K

^wxCDAudioLinux::~~wxCDAudioLinux

^wxcdaudiolinuxdtor

^browse00009

^K wxCDAudioLinux ~wxCDAudioLinux

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`xcdaudiolinux')")

^K ~wxCDAudioLinux

^{\$#+K!}**wxCDAudioLinux::Play**

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

^wxCDAudioLinux::Play

^wxcdaudiolinuxplay

^browse00010

^K wxCDAudioLinux Play

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`xcdaudiolinux')")

^K Play

^{\$#+K!}**wxCDAudioLinux::Pause**

bool Pause()^K

^wxCDAudioLinux::Pause

^wxcdaudiolinuxpause

^browse00011

^K wxCDAudioLinux Pause

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`xcdaudiolinux')")

^K Pause

`$#+K!wxCDAudioLinux::Resume`

`bool Resume()`^K

^w`xCDAudioLinux::Resume`

^w`xcdaudiolinuxresume`

^b`rowse00012`

^K `wxCDAudioLinux Resume`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`xcdaudiolinux')")`

^K `Resume`

wxCDAudioLinux::GetStatus

CDstatus GetStatus()

wxCDAudioLinux::GetStatus

wxcdaudiolinuxgetstatus

rowse00013

wxCDAudioLinux GetStatus

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(mmedia.hlp',
`wxcdaudiolinux')")**

GetStatus

^{\$#+K!}**wxCDAudioLinux::GetTime**

wxCDtime GetTime()^K

^wxCDAudioLinux::GetTime

^wxcdaudiolinuxgettime

^browse00014

^K wxCDAudioLinux GetTime

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`xcdaudiolinux')")

^K GetTime

\$#+K!wxCDAudioLinux::GetToc

CDtoc& GetToc()^K

^wxCDAudioLinux::GetToc

^wxcdaudiolinuxgettoc

^browse00015

^K wxCDAudioLinux GetToc

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`xcdaudiolinux')")

^K GetToc

\$#+KK!**wxCDAudioLinux::Ok**

bool Ok() const

wxCDAudioLinux::Ok

wxcdaudiolinuxok

rowse00016

wxCDAudioLinux Ok

Ok

**EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxcdaudiolinux')")**

^{\$#+K!}**wxCDAudioLinux::OpenDevice**

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

^wxCDAudioLinux::OpenDevice

^wxcdaudiolinuxopendevic

^browse00017

^K wxCDAudioLinux OpenDevice

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`xcdaudiolinux')")

^K OpenDevice

wxCDAudioWin::wxCDAudioWin

wxCDAudioWin()^K

wxCDAudioWin(const char* dev_name)^K

wxCDAudioWin::wxCDAudioWin

wxcdaudiowinwxcdaudiowin

rowse00019

wxCDAudioWin wxCDAudioWin

**nableButton("Up");ChangeButtonBinding("Up", "JumpId(mmedia.hlp',
`wxcdaudiowin')")**

wxCDAudioWin

wxCDAudioWin

^{\$#+K!}**wxCDAudioWin::~~wxCDAudioWin**

~wxCDAudioWin()^K

^w**wxCDAudioWin::~~wxCDAudioWin**

^w**wxcdaudiowindtor**

^b**rowse00020**

^K **wxCDAudioWin ~wxCDAudioWin**

^E**nableButton("Up");ChangeButtonBinding("Up", "JumpId(\ mmedia.hlp',
`wxcdaudiowin')")**

^K **~wxCDAudioWin**

^{\$#+K!}**wxCDAudioWin::Play**

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

^wxCDAudioWin::Play

^wxcdaudiowinplay

^browse00021

^K wxCDAudioWin Play

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',

[`]wxcdaudiowin')")

^K Play

\$#+K! wxCDAudioWin::Pause

bool Pause()^K

^wxCDAudioWin::Pause

^wxcdaudiowinpause

^browse00022

^K wxCDAudioWin Pause

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`xcdaudiowin')")

^K Pause

`$#+K!wxCDAudioWin::Resume`

`bool Resume()`^K

^w`xCDAudioWin::Resume`

^w`xcdaudiowinresume`

^b`rowse00023`

^K `wxCDAudioWin Resume`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(\mmedia.hlp',
`xcdaudiowin')")`

^K `Resume`

\$#+K! **wxCDAudioWin::GetStatus**

CDstatus GetStatus()K

wxCDAudioWin::GetStatus
wxcdaudiowingetstatus
browse00024
K wxCDAudioWin GetStatus
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\mmedia.hlp',
`wxcdaudiowin')")
K GetStatus

^{\$#+K!}**wxCDAudioWin::GetTime**

wxCDtime GetTime()^K

^wxCDAudioWin::GetTime

^wxcdaudiowingettextime

^browse00025

^K wxCDAudioWin GetTime

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\mmedia.hlp',
`xcdaudiowin')")

^K GetTime

^{\$#+K!}**wxCDAudioWin::GetToc**

const CDtoc& GetToc()^K

^wxCDAudioWin::GetToc

^wxcdaudiowingettoc

^browse00026

^K wxCDAudioWin GetToc

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\mmedia.hlp',
`xcdaudiowin')")

^K GetToc

\$#+KK!**wxCDAudioWin::Ok**

bool Ok() const

^wxCDAudioWin::Ok

^wxcdaudiowinok

^browse00027

^K wxCDAudioWin Ok

^K Ok

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxcdaudiowin')")

\$#+K!wxCDAudioWin::PrepareToc

void PrepareToc()^K

^wxCDAudioWin::PrepareToc

^wxcdaudiowinpreparetoc

^browse00028

^K wxCDAudioWin PrepareToc

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp",

`xcdaudiowin')")

^K PrepareToc

^{\$#+K!}**CDtoc::CDtoc**

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

^CDtoc::CDtoc

^cdtoccdtoc

^browse00030

^K CDtoc CDtoc

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `cdtoc')")

^K CDtoc

\$#+KK!^CDtoc::GetTrackTime

wxCDtime GetTrackTime(wxUint8 *track*) const

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

^CDtoc::GetTrackTime

^cdtocgettracktime

^browse00031

^K CDtoc GetTrackTime

^K GetTrackTime

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp`, `cdtoc`)")

CDtoc::GetTrackPos

wxCDtime GetTrackPos(wxUint8 *track*) const

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

CDtoc::GetTrackPos

CDtocGetTrackPos

CDtocGetTrackPos

CDtoc GetTrackPos

GetTrackPos

EnableButton("Up");ChangeButtonBinding("Up", "JumpId('mmedia.hlp', 'cdtoc')")

\$#+KK!^CDtoc::GetTotalTime

wxCDtime GetTotalTime() const

Returns the total time

^CDtoc::GetTotalTime

^cdtocgettotaltime

^browse00033

^K CDtoc GetTotalTime

^K GetTotalTime

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp`, `cdtoc`)")

```
$#+K!wxCDAudio::wxCDAudio
wxCDAudio()K
```

```
wxCDAudio::wxCDAudio
wxcdaudiowxcdaudio
browse00034
K wxCDAudio wxCDAudio
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp', `cdtoc')")
K wxCDAudio
```

\$#+K!wxCDAudio::~~wxCDAudio

~wxCDAudio()^K

^wxCDAudio::~~wxCDAudio

^wxcdaudiotor

^browse00035

^K wxCDAudio ~wxCDAudio

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `cdtoc')")

^K ~wxCDAudio

wxCDAudio::Play

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

Play audio at the specified position

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

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

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

^wxCDAudio::Play

^wxcdaudioplay

^browse00036

^K wxCDAudio Play

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('mmedia.hlp', `cdtoc')")

^K Play

^K Play

^K Play

`$#+K!wxCDAudio::Pause`

`bool Pause()K`

Pause the audio playing

`wxCDAudio::Pause`

`wxcdaudiopause`

`browse00037`

`KwxCDAudio Pause`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp', `cdtoc')")`

`KPause`

`$#+K!wxCDAudio::Resume`

`bool Resume()`^K

Resume a paused audio playing

^w`xCDAudio::Resume`

^w`xcdaudioresume`

^b`rowse00038`

^K`wxCDAudio Resume`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `cdtoc')")`

^K`Resume`

wxCDAudio::GetStatus

CDstatus GetStatus()

Get the current CD status

wxCDAudio::GetStatus

wxcdaudiogetstatus

rowse00039

wxCDAudio GetStatus

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `cdtoc')")

GetStatus

\$#+K! wxCDAudio::GetTime

wxCDtime GetTime()^K

Get the current playing time

^wxCDAudio::GetTime

^wxcdaudiogettime

^browse00040

^K wxCDAudio GetTime

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `cdtoc')")

^K GetTime

\$#+K! **wxCDAudio::GetToc**

const CDtoc& GetToc()^K

Returns the table of contents

wxCDAudio::GetToc

wxcdaudiogettoc

browse00041

K wxCDAudio GetToc

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `cdtoc')")

K GetToc

\$#+KK!**wxCDAudio::Ok**

bool Ok() const

CD ok

^wxCDAudio::Ok

^wxcdaudiook

^browse00042

^K wxCDAudio Ok

^K Ok

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp', `cdtoc')")

`$#+K! wxSoundStream::wxSoundStream`

`wxSoundStream()`^K

Default constructor.

`wxSoundStream::wxSoundStream`

`wxsoundstreamwxsoundstream`

`browse00044`

`K wxSoundStream wxSoundStream`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundstream')")`

`K wxSoundStream`

`$#+K! wxSoundStream::~~wxSoundStream`

`~wxSoundStream()`^K

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

`wxSoundStream::~~wxSoundStream`

`wxsoundstreamdctor`

`browse00045`

`K wxSoundStream ~wxSoundStream`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\media.hlp',
`wxsoundstream')")`

`K ~wxSoundStream`

wxSoundStream::Read

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

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](#).

It is better to use the size returned by [wxSoundStream::GetBestSize](#): 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](#), [wxSoundStream::GetSoundFormat](#).

data

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

^wxSoundStream::Read

^wxsoundstreamread

^browse00046

^K wxSoundStream Read

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundstream')")

^K Read

^{\$#+K!}**wxSoundStream::Write**

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

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](#).

It is better to use the size returned by [wxSoundStream::GetBestSize](#): 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](#), [wxSoundStream::GetSoundFormat](#).

buffer

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

^wxSoundStream::Write

^wxsoundstreamwrite

^browse00047

^K wxSoundStream Write

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\mmedia.hlp',
`wxsoundstream')")

^K Write

`$#+KK!` **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::GetBestSize
wxsoundstreamgetbestsize
browse00048
K wxSoundStream GetBestSize
K GetBestSize
E enableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundstream')")
```

^{S#+K!}**wxSoundStream::SetSoundFormat**

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

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](#). See [wxSoundStream errors section](#) 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](#).

^wxSoundStream::SetSoundFormat

^wxsoundstreamsetsoundformat

^browse00049

^K wxSoundStream SetSoundFormat

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\ mmedia.hlp',
`wxsoundstream'))

^K SetSoundFormat

`$#+KK!` **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::GetSoundFormat
wxsoundstreamgetsoundformat
rowse00050
K wxSoundStream GetSoundFormat
K GetSoundFormat
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundstream')")
```

^{\$#+K!}**wxSoundStream::SetCallback**

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

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::SetCallback

^wxsoundstreamregister

^browse00051

^K wxSoundStream SetCallback

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundstream'))

^K Register

`$#+K!` **wxSoundStream::StartProduction**

bool StartProduction(int evt)^K

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 automatically 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::StartProduction

^wxsoundstreamstartproduction

^browse00052

^K wxSoundStream StartProduction

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundstream'))

^K StartProduction

wxSoundStream::StopProduction

bool StopProduction()

It stops the async notifier and the sound streaming straightly.

wxSoundStream::StopProduction

wxSoundStream::StopProduction

bool StopProduction

wxSoundStream::StopProduction

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxSoundStream')")**

StopProduction

\$#+K! **wxSoundStream::SetEventHandler**

void SetEventHandler(wxSoundStream* *handler*)^K

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

WxSoundStream::SetEventHandler

Wxsoundstreamseteventhandler

browse00054

K wxSoundStream SetEventHandler

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundstream'))

K SetEventHandler

`wxSoundStream::GetError`

`wxSoundError GetError() const`

It returns the last error which occurred.

`wxSoundStream::GetError`

`wxsoundstreamgeterror`

`rowse00055`

`wxSoundStream GetError`

`GetError`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundstream')")`

`$#+KK!` **wxSoundStream::GetLastAccess**

wxUInt32 GetLastAccess() const

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

`w`xSoundStream::GetLastAccess

`w`xsoundstreamgetlastaccess

`b`rowse00056

`K` wxSoundStream GetLastAccess

`K` GetLastAccess

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundstream')")

`$#+KK!` **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::QueueFilled
wxsoundstreamqueuefilled
browse00057
K wxSoundStream QueueFilled
K QueueFilled
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundstream')")
```

wxSoundStream::OnSoundEvent

void OnSoundEvent(int evt)

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

wxSoundStream::OnSoundEvent

wxsoundstreamonsoundevent

rowse00058

wxSoundStream OnSoundEvent

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundstream'))**

OnSoundEvent

wxSoundFileStream::wxSoundFileStream

wxSoundFileStream(wxInputStream& stream, wxSoundStream& io_sound)^K

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)^K

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.

^wwxSoundFileStream::wxSoundFileStream

^wxsoundfilestreamwxsoundfilestream

^browse00060

^K wxSoundFileStream wxSoundFileStream

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(mmedia.hlp',
`xsoundfilestream')")

^K wxSoundFileStream

^K wxSoundFileStream

\$#+K! **wxSoundFileStream::~~wxSoundFileStream**

~wxSoundFileStream()^K

It destroys the current sound file codec.

wxSoundFileStream::~~wxSoundFileStream

wxsoundfilestreamdtor

browse00061

K wxSoundFileStream ~wxSoundFileStream

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundfilestream')")

K ~wxSoundFileStream

`$#+K!wxSoundFileStream::Play`

`bool Play()`^K

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::Play`

`wxsoundfilestreamplay`

`browse00062`

`K wxSoundFileStream Play`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\mmedia.hlp',
`wxsoundfilestream')")`

`K Play`

wxSoundFileStream::Record

bool Record(wxUint32 *time*)^K

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), you can set it to `wxSOUND_INFINITE_TIME`.

On success, it returns TRUE.

^wwxSoundFileStream::Record

^wxsoundfilestreamrecord

^browse00063

^K wxSoundFileStream Record

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`xsoundfilestream')")

^K Record

\$#+K! **wxSoundFileStream::Stop**

bool Stop()^K

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::Stop

wxsoundfilestreamstop

browse00064

K wxSoundFileStream Stop

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\ mmedia.hlp',
`wxsoundfilestream')")

K Stop

\$#+K! **wxSoundFileStream::Pause**

bool Pause()^K

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 TRUE.

wxSoundFileStream::Pause

wxsoundfilestreampause

browse00065

K wxSoundFileStream Pause

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\mmmedia.hlp',
`wxsoundfilestream')")

K Pause

\$#+K! **wxSoundFileStream::Resume**

bool Resume()^K

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

^wwxSoundFileStream::Resume
^wwxsoundfilestreamresume
^browse00066
^K wxSoundFileStream Resume
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundfilestream')")
^K Resume

`$#+KK!`**wxSoundFileStream::IsStopped**

bool IsStopped() const

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

`w`xSoundFileStream::IsStopped
`w`xsoundfilestreamisstopped
`b`rowse00067
`K` wxSoundFileStream IsStopped
`K` IsStopped
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wsoundfilestream`")")

`$#+KK!` **wxSoundFileStream::IsPaused**

bool IsPaused() const

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

`w`xSoundFileStream::IsPaused

`w`xsoundfilestreamispaused

`b`rowse00068

`K` wxSoundFileStream IsPaused

`K` IsPaused

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundfilestream`")

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::StartProduction

wxsoundfilestreamstartproduction

rowse00069

wxSoundFileStream StartProduction

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(\ mmedia.hlp',
`wxsoundfilestream')")**

StartProduction

`wxSoundFileStream::StopProduction`

`bool StopProduction()`^K

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

^w`wxSoundFileStream::StopProduction`

^w`wxsoundfilestreamstopproduction`

^b`rowse00070`

^K `wxSoundFileStream StopProduction`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundfilestream')")`

^K `StopProduction`

\$#+K! **wxSoundFileStream::GetLength**

wxUInt32 GetLength()^K

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](#) and [GetTimeFromBytes](#).

^wxSoundFileStream::GetLength
^wxsoundfilestreamgetlength
^browse00071
^K wxSoundFileStream GetLength
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundfilestream')")
^K GetLength

\$#+K! **wxSoundFileStream::GetPosition**

wxUInt32 GetPosition()^K

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](#) and [GetTimeFromBytes](#).

WxSoundFileStream::GetPosition

Wxsoundfilestreamgetposition

browse00072

K wxSoundFileStream GetPosition

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\ mmedia.hlp',
`wxsoundfilestream')")

K GetPosition

^{\$#+K!}**wxSoundFileStream::SetPosition**

wxUInt32 SetPosition(wxUInt32 *new_position*)^K

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](#) and [GetTimeFromBytes](#).

On success, it returns TRUE.

Warning

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

^wxSoundFileStream::SetPosition

^wxsoundfilestreamsetposition

^browse00073

^K wxSoundFileStream SetPosition

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundfilestream')")

^K SetPosition

\$#+K! **wxSoundFileStream::Read**

wxSoundStream& Read(**void*** *buffer*, **wxUInt32** *len*)^K

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::Read

Wxsoundfilestreamread

browse00074

K wxSoundFileStream Read

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundfilestream')")

K Read

^{\$#+K!}**wxSoundFileStream::Write**

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

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::Write

^wxsoundfilestreamwrite

^browse00075

^K wxSoundFileStream Write

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundfilestream')")

^K Write

^{\$#+K!}**wxSoundFileStream::SetSoundFormat**

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

^wxSoundFileStream::SetSoundFormat

^wxsoundfilestreamsetsoundformat

^browse00076

^K wxSoundFileStream SetSoundFormat

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundfilestream')")

^K SetSoundFormat

`$#+KK!` **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::GetCodecName
wxsoundfilestreamgetcodecname
rowse00077
K wxSoundFileStream GetCodecName
K GetCodecName
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundfilestream`"))
```

`$#+K!wxSoundFileStream::CanRead`

`bool CanRead()`^K

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

`wxSoundFileStream::CanRead`

`wxsoundfilestreamcanread`

`browse00078`

`K wxSoundFileStream CanRead`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundfilestream')")`

`K CanRead`

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::PrepareToPlay`

`wxsoundfilestreampreparetoplay`

`rowse00079`

`wxSoundFileStream PrepareToPlay`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(\media.hlp',
`wxsoundfilestream')")`

`PrepareToPlay`

^{\$#+K!}**wxSoundFileStream::PrepareToRecord**

bool PrepareToRecord(wxUint32 *time*)^K

^wxSoundFileStream::PrepareToRecord

^wxsoundfilestreampreparetorecord

^browse00080

^K wxSoundFileStream PrepareToRecord

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundfilestream')")

^K PrepareToRecord

wxSoundFileStream::FinishRecording

bool FinishRecording()

wxSoundFileStream::FinishRecording

wxsoundfilestreamfinishrecording

rowse00081

wxSoundFileStream FinishRecording

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(\ mmedia.hlp',
`wxsoundfilestream')")**

FinishRecording

^{\$#+K!}**wxSoundFileStream::RepositionStream**

bool RepositionStream(wxUint32 *position*)^K

This is called by wxSoundFileStream::SetPosition to seek the input stream to the right position. This must be overridden 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::RepositionStream

^wxsoundfilestreamrepositionstream

^browse00082

^K wxSoundFileStream RepositionStream

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundfilestream')")

^K RepositionStream

`$#+K!` **wxSoundFileStream::FinishPreparation**

void FinishPreparation(wxUInt32 *len*)^K

This is an internal function but it must be 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`.

^wwxSoundFileStream::FinishPreparation

^wxsoundfilestreamfinishpreparation

^browse00083

^K wxSoundFileStream FinishPreparation

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(\`mmedia.hlp',
`wxsoundfilestream')")

^K FinishPreparation

^{\$#+K!}**wxSoundFileStream::GetData**

wxUInt32 **GetData**(**void*** *buffer*, **wxUInt32** *len*)^K

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 overridden by the file codec class. The default behaviour is simply to read from the input stream.

^wxSoundFileStream::GetData

^wxsoundfilestreamgetdata

^browse00084

^K wxSoundFileStream GetData

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wsoundfilestream')")

^K GetData

^{\$#+K!}**wxSoundFileStream::PutData**

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

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 overridden by the file codec class. The default behaviour is simply to write to the input stream.

^wxSoundFileStream::PutData

^wxsoundfilestreamputdata

^browse00085

^K wxSoundFileStream PutData

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundfilestream')")

^K PutData

wxSoundFormatBase::wxSoundFormatBase

wxSoundFormatBase()

wxSoundFormatBase::wxSoundFormatBase

wxsoundformatbasewxsoundformatbase

rowse00087

wxSoundFormatBase wxSoundFormatBase

enableButton("Up");ChangeButtonBinding("Up", "JumpId(\ mmedia.hlp',

`wsoundformatbase')

wxSoundFormatBase

^{\$#+K!}**wxSoundFormatBase::~~wxSoundFormatBase**

~wxSoundFormatBase()^K

^wxSoundFormatBase::~~wxSoundFormatBase

^wxsoundformatbasedtor

^browse00088

^K wxSoundFormatBase ~wxSoundFormatBase

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^mmedia.hlp',
`wxsoundformatbase')")

^K ~wxSoundFormatBase

`$#+KK!` **wxSoundFormatBase::GetType**

wxSoundFormatType GetType() const

It returns a "standard" format type.

`w`xSoundFormatBase::GetType

`w`xsoundformatbasegettype

`b`rowse00089

`K` wxSoundFormatBase GetType

`K` GetType

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wsoundformatbase')")

`wxSoundFormatBase::Clone`

`wxSoundFormatBase* Clone() const`

It clones the current format.

`wxSoundFormatBase::Clone`

`wxsoundformatbaseclone`

`rowse00090`

`wxSoundFormatBase Clone`

`Clone`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wsoundformatbase')")`

`$#+KK!wxSoundFormatBase::GetTimeFromBytes`

`wxUint32 GetTimeFromBytes(wxUint32 bytes) const`

`wxSoundFormatBase::GetTimeFromBytes`

`wxsoundformatbasegettimefrombytes`

`browse00091`

`K wxSoundFormatBase GetTimeFromBytes`

`K GetTimeFromBytes`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundformatbase')")`

\$#+KK! **wxSoundFormatBase::GetBytesFromTime**

wxUint32 GetBytesFromTime(wxUint32 *time*) const

wxSoundFormatBase::GetBytesFromTime

wxsoundformatbasegetbytesfromtime

browse00092

K wxSoundFormatBase GetBytesFromTime

K GetBytesFromTime

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundformatbase`)")

`$#+KK!wxSoundFormatBase::operator!=`

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

`wxSoundFormatBase::operator!=
wxsoundformatbaseoperatomeq
browse00093
K wxSoundFormatBase operator!=
K operator, =
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`mmedia.hlp',
`wxsoundformatbase')")`

