MIDAS Sound System

API Reference

Petteri Kangaslampi

January 27, 1997

Contents

Chapter 1

Introduction

1.1 About this document

This document contains a full programmer's reference for the MIDAS Application Programming interface. It includes complete descriptions of all constants, data structure and functions available in the API, plus examples on how to use them. It is not intended to be a tutorial on using MIDAS — for that kind of information see MIDAS Programmer's Guide.

1.2 Document organization

The document itself is divided into five different chapters, according to different functional groups. In addition to this introduction, the chapters cover configuration, initialization and overall system control, module playback, sample playback and stream playback. Each chapter is further divided into three sections: constants, data types and functions.

Chapter 2

Configuration, initialization and control

2.1 Constants

This section describes all constants used in MIDAS initialization and configuration. They are grouped according to the `enum` used to define them.

2.1.1 MIDASoptions

`enum MIDASoptions`

Description

These constants are used with the function *MIDASsetOption* to change different MIDAS configuration options.

Values

MIDAS_OPTION_MIXRATE Output mixing rate

MIDAS_OPTION_OUTPUTMODE Output mode, see `enum` *MIDASmodes*

MIDAS_OPTION_MIXBUFLEN Mixing buffer length, in milliseconds

MIDAS_OPTION_MIXBUFBLOCKS The number of blocks the buffer should be divided into

2.1.2 MIDASmodes

```
enum MIDASmodes
```

Description

These constants are used to describe different MIDAS output modes. They are used with the function *MIDASsetOption*, when changing the setting *MIDAS_OPTION_OUTPUTMODE*.

Values

MIDAS_MODE_8BIT_MONO 8-bit mono output

MIDAS_MODE_16BIT_MONO 16-bit mono output

MIDAS_MODE_8BIT_STEREO 8-bit stereo output

MIDAS_MODE_16BIT_STEREO 16-bit stereo output

2.2 Data types

This section describes all data types used in MIDAS initialization and configuration.

2.3 Functions

This section describes all functions available for MIDAS initialization, configuration and system control. This includes error handling.

2.3.1 MIDASstartup

```
BOOL MIDASstartup(void)
```

Prepares MIDAS Sound System for initialization and use.

Input

None.

Description

This function sets all MIDAS configuration variables to default values and prepares MIDAS for use. It must be called before any other MIDAS function, including *MIDASinit* and *MIDASsetOption*, is called. After this function has been called, *MIDASclose* can be safely called at any point and any number of times, regardless of whether MIDAS has been initialized or not. After calling this function, you can use *MIDASsetOption* to change MIDAS configuration before initializing MIDAS with *MIDASinit*.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASsetOption, *MIDASinit*, *MIDASclose*

2.3.2 MIDASgetLastError

```
int MIDASgetLastError(void)
```

Gets the MIDAS error code for last error.

Input

None.

Description

This function can be used to read the error code for most recent failure. When a MIDAS API function returns an error condition, this function can be used to determine the actual cause of the error, and this error can then be reported to the user or ignored, depending on the kind of response needed. Use *MIDASgetErrorMessage* to get a textual message corresponding to an error code.

This function can be called at any point after *MIDASstartup* has been called.

Return value

MIDAS error code for the most recent error.

Operating systems

All

See also

MIDASgetErrorMessage

2.3.3 MIDASgetErrorMessage

```
char *MIDASgetErrorMessage(int errorCode)
```

Gets an error message corresponding to an error code.

Input

errorCode The error code from *MIDASgetLastError*

Description

This function returns a textual error message corresponding to a MIDAS error code. It can be used for displaying an error message to the user. Use *MIDASgetLastError* to determine the error code.

This function can be called at any point after *MIDASstartup* has been called.

Return value

Error message string corresponding to the error code.

Operating systems

All

See also

MIDASgetLastError

2.3.4 MIDASinit

```
BOOL MIDASinit(void)
```

Initializes MIDAS Sound System.

Input

None.

Description

This function initializes all MIDAS Sound System components, and sets up the API. Apart from configuration functions, this function must be called before any other MIDAS functions are used.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASsetOption, *MIDASclose*

2.3.5 MIDASclose

```
BOOL MIDASclose(void)
```

Uninitializes MIDAS Sound System.

Input

None.

Description

This function uninitializes all MIDAS Sound System components, deallocates all resources allocated, and shuts down all MIDAS processing. This function must always be called before exiting under MS-DOS and is also strongly recommended under other operating systems. After this function has been called, no MIDAS function may be called unless MIDAS is initialized again.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASinit

2.3.6 MIDASdetectSD

```
BOOL MIDASdetectSD(void)
```

Attempts to detect the Sound Device to use.

Input

None.

Description

This function attempts to detect the Sound Device that should be used. It is called internally by *MIDASinit*, and should not normally be called by user programs. Note that the function returns TRUE even if it fails to detect a Sound Device.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASinit

2.3.7 MIDASsetOption

`BOOL MIDASsetOption(int option, int value)`

Sets a MIDAS option.

Input

option Option number (see `enum` *MIDASoptions* above)

value New value for option

Description

This function sets a value to a MIDAS option. The different number codes for different options are described above. All MIDAS configuration options must be set with this function **before** *MIDASinit* is called.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASinit

2.3.8 MIDASopenChannels

```
BOOL MIDASopenChannels(int numChans)
```

Opens Sound Device channels for sound and music output.

Input

numChans Number of channels to open

Description

This function opens a specified number of channels for digital sound and music output. The channels opened can be used for playing streams, samples and modules. When *MIDASplayModule* is used to play modules, it will use the last possible channels for the module, so that the first `(numChans - number-of-channels-in-module)` channels are available for effects and streams.

If this function has not been called before *MIDASplayModule* is called, *MIDASplayModule* will open the channels it needs for module playback. However, if this function has been called, but the number of channels opened is inadequate for the module, *MIDASplayModule* will return an error and refuse to play the module.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDAScloseChannels, *MIDASplayModule*

2.3.9 MIDAScloseChannels

```
BOOL MIDAScloseChannels(void)
```

Closes Sound Device channels opened with *MIDASopenChannels*.

Input

None.

Description

This function closes Sound Device channels that were opened with *MIDASopenChannels*. Note that you may **not** use this function to close channels that were opened by *MIDASplayModule* — *MIDASstopModule* will do that automatically.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASopenChannels, *MIDASplayModule*, *MIDASstopModule*

2.3.10 MIDASstartBackgroundPlay

```
BOOL MIDASstartBackgroundPlay(DWORD pollRate)
```

Starts playing music and sound in the background.

Input

pollRate Polling rate (number of polls per second), 0 for default

Description

This function starts playing sound and music in the background. **pollRate** controls the target polling rate — number of polls per second. Polling might not be done at actually the set rate, although usually it will be faster. Under Win32 and Linux, a new thread will be created for playing. **Under MS-DOS this function is currently ignored, and background playback starts immediately when MIDAS is initialized.**

Return value

TRUE if successful, FALSE if not.

Operating systems

All, but see MS-DOS note above.

See also

MIDASstopBackgroundPlay, *MIDASpoll*

2.3.11 MIDASstopBackgroundPlay

```
BOOL MIDASstopBackgroundPlay(void)
```

Stops playing sound and music in the background.

Input

None.

Description

This function stops music and sound background playback that has been started with *MIDASstartBackgroundPlay*. Under Win32 and Linux, this function also destroys the thread created for playback. **Under MS-DOS this function is currently ignored, and background playback starts immediately when MIDAS is initialized.**

If background playback has been started with *MIDASstartBackgroundPlay*, this function bf must be called before the program exits.

Return value

TRUE if successful, FALSE if not.

Operating systems

All, but see MS-DOS note above.

See also

MIDASstartBackgroundPlay, *MIDASpoll*

2.3.12 MIDASpoll

```
BOOL MIDASpoll(void)
```

Polls the MIDAS sound and music player.

Input

None.

Description

This function can be used to poll MIDAS sound and music player manually. It plays music forward, mixes sound data, and sends it to output. When using manual polling, make sure you call *MIDASpoll* often enough to make sure there are no breaks in sound output — at least two times during buffer length, preferably four times or more. Under multitasking operating systems such as Win32 and Linux, this may be difficult, so very short buffer sizes can't be used reliably.

Also note that **currently this function is not available under MS-DOS.** Under MS-DOS, playback is always done in background using the system timer (IRQ 0).

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32, Linux

See also

MIDASstartBackgroundPlay, *MIDASstopBackgroundPlay*

2.3.13 MIDASgetVersionString

```
char *MIDASgetVersionString(void)
```

Gets the current MIDAS version as a string.

Input

None.

Description

This function can be used to determine the MIDAS version being loaded. It returns a text string description of the version. Version numbers are usually of form “x.y.z”, where “x” is the major version number, “y” minor version number and “z” patch level. In some occasions, “z” can be replaced with a textual message such as “rc1” for Release Candidate 1. All MIDAS versions with the major and minor version numbers equal have a compatible DLL API, and can be used interchangeably.

Return value

MIDAS Sound System version number as a string.

Operating systems

Win32, Linux

See also

Chapter 3

Module playback

3.1 Constants

This section describes all constants used in MIDAS module playback. They are grouped according to the `enum` used to define them.

3.2 Data types

This section describes all data types used in MIDAS module playback.

3.2.1 MIDASmoduleInfo

```
typedef struct
{
    char        songName[32];
    unsigned    songLength;
    unsigned    numPatterns;
    unsigned    numInstruments;
    unsigned    numChannels;
} MIDASmoduleInfo;
```

Module information structure.

Members

songName Module song name, an ASCIIZ string

songLength Module song length in number of positions

numPatterns Number of patterns in module

numInstruments Number of instruments in module

numChannels The number of channels the module uses

Description

This structure is used with the function *MIDASgetModuleInfo* to query information about an module. *MIDASgetModuleInfo* fills a *MIDASmoduleInfo* structure with the information.

3.2.2 MIDASinstrumentInfo

```
typedef struct
{
    char        instName[32];
} MIDASinstrumentInfo;
```

Instrument information structure.

Members

instName Instrument name, an ASCIIZ string

Description

This structure is used with the function *MIDASgetInstrumentInfo* to query information about an instrument in a module. *MIDASgetInstrumentInfo* fills a *MIDASinstrumentInfo* structure with the information.

3.2.3 MIDASplayStatus

```
typedef struct
{
    unsigned    position;
    unsigned    pattern;
    unsigned    row;
    int         syncInfo;
} MIDASplayStatus;
```

Module status information structure.

Members

position Current playback position number

pattern Current playback pattern number

row Current playback row number

syncInfo Latest synchronization command infobyte, -1 if no synchronization command has been encountered yet.

Description

This structure is used with the function *MIDASgetPlayStatus* to query the current module playback status. *MIDASgetPlayStatus* fills a *MIDASplayStatus* structure with the information.

Some more information about the synchronization commands: In protracker and Scream Tracker 3 modules, the command `Wxx` is used as a music synchronization command. The infobyte of this command is available as the music synchronization command infobyte above.

3.2.4 MIDASmodule

```
typedef ... MIDASmodule;
```

Description

MIDASmodule is a module handle that defines a module that has been loaded into memory.

3.3 Functions

This section describes all functions available for MIDAS module playback.

3.3.1 MIDASloadModule

```
MIDASmodule MIDASloadModule(char *fileName)
```

Loads a module file into memory.

Input

fileName Module file name

Description

This function loads a module file into memory. It checks the module format based on the module file header, and invokes the correct loader to load the module into memory in GMPlayer internal format. The module can then be played using *MIDASplayModule*, and deallocated from memory with *MIDASfreeModule*.

Return value

Module handle if successful, NULL if not.

Operating systems

All

See also

MIDASplayModule, *MIDASfreeModule*

3.3.2 MIDASplayModule

```
BOOL MIDASplayModule(MIDASmodule module, int numEffectChannels)
```

Starts playing a module.

Input

module Module to be played

numEffectChannels Number of sound channels to leave open for effects

Description

This functions starts playing a module that has been previously loaded with *MIDASloadModule*. If channels have not been previously opened using *MIDASopenChannels*, this function opens the channels necessary to play the module, and if **numEffectChannels** is nonzero, it opens additional channels for sound effects. The module is always played on the last possible channels, so the first **numEffectChannels** are available for effects and streams.

Note! Currently only one module can be played at a time.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASloadModule, *MIDASstopModule*, *MIDASopenChannels*

MIDASstopModule

```
BOOL MIDASstopModule(MIDASmodule module)
```

Stops playing a module.

Input

module Module that is being played

Description

This function stops playing a module that has been played with *MIDASplay-Modyle*. If the channels were opened automatically by *MIDASplayModule*, this function will close them, but if they were opened manually with *MIDA-SopenChannels*, they will be left open.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplayModule, *MIDASopenChannels*

3.3.3 MIDASfreeModule

```
BOOL MIDASfreeModule(MIDASmodule module)
```

Deallocates a module.

Input

module Module that should be deallocated

Description

This function deallocates a module loaded with *MIDASloadModule*. It should be called to free unused modules from memory, after they are no longer being played, to avoid memory leaks.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASloadModule

3.3.4 MIDASgetPlayStatus

```
BOOL MIDASgetPlayStatus(MIDASplayStatus *status)
```

Gets module playback status.

Input

status Pointer to playback status structure where the status will be written.

Description

This function reads the current module playback status, and writes it to ***status**. The user needs to pass it a valid pointer to a *MIDASplayStatus* structure, which will be updated.

Return value

TRUE if successful, FALSE if not. The current playback status is written to ***status**.

Operating systems

All

See also

MIDASplayModule, *MIDASplayStatus*

3.3.5 MIDASsetPosition

```
BOOL MIDASsetPosition(int newPosition)
```

Changes module playback position.

Input

newPosition New playback position

Description

This function changes the current module playback position. The song starts at position 0, and the length is available in the *MIDASmoduleInfo* structure. You should make sure that **position** lies inside those limits. To skip backward or forward a single position, first read the current position with *MIDASgetPlayStatus*, and substract or add one to the current position.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplayModule, *MIDASgetPlayStatus*, *MIDASgetModuleInfo*

3.3.6 MIDASsetMusicVolume

```
BOOL MIDASsetMusicVolume(unsigned volume)
```

Changes music playback volume.

Input

volume New music playback volume (0–64)

Description

This function changes the music playback master volume. It can be used, for example, for fading music in or out smoothly, or for adjusting the music volume relative to sound effects. The volume change only affects the song that is currently being played — if a new song is started, the volume is reset. The default music volume is 64 (the maximum).

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

3.3.7 MIDASgetModuleInfo

```
BOOL MIDASgetModuleInfo(MIDASmodule module,
    MIDASmoduleInfo *info)
```

Gets information about a module.

Input

module Module handle for the module

info Pointer to a module info structure where the information willl be written

Description

This function returns information about a module, including the module name and the number of channels used. The user needs to pass it a valid pointer to a *MIDASmoduleInfo* structure (***info**), where the information will be written.

Return value

TRUE if successful, FALSE if not. The module information is written to ***info**.

Operating systems

All

See also

MIDASplayModule, *MIDASmoduleInfo*

3.3.8 MIDASgetInstrumentInfo

```
BOOL MIDASgetInstrumentInfo(MIDASmodule module,
    int instNum, MIDASinstrumentInfo *info)
```

Gets information about an instrument in a module.

Input

module Module handle for the module

instNum Instrument number

info Pointer to an instrument info structure where the information willl be written

Description

This function returns information about an instrument in a module, including the instrument name. The user needs to pass it a valid pointer to a *MIDASinstrumentInfo* structure (***info**), where the information will be written. You should ensure that **instNum** is a valid instrument number. Instrument numbers start from 0, although trackers traditionally number them from 1, and you can use*MIDASgetModuleInfo* to get the number of instruments available in a module.

Return value

TRUE if successful, FALSE if not. The instrument information is written to ***info**.

Operating systems

All

See also

MIDASplayModule, *MIDASgetModuleInfo*, *MIDASmoduleInfo*

Chapter 4

Sample playback

4.1 Constants

This section describes all constants used in MIDAS sample playback. They are grouped according to the `enum` used to define them.

4.1.1 MIDASsampleTypes

`enum MIDASsampleTypes`

Description

These constants identify different sample types. They are used with the functions *MIDASloadRawSample*, *MIDASplayStreamFile* and *MIDASplayStreamPolling* to indicate the format of the sample data. The byte order of the sample data is always the system native order (little endian for Intel x86 systems).

Values

MIDAS_SAMPLE_8BIT_MONO 8-bit mono sample, unsigned

MIDAS_SAMPLE_8BIT_STEREO 8-bit stereo sample, unsigned

MIDAS_SAMPLE_16BIT_MONO 16-bit mono sample, signed

MIDAS_SAMPLE_16BIT_STEREO 16-bit stereo sample, signed

4.1.2 MIDASloop

```
enum MIDASloop
```

Description

These constants are used to indicate the loop type of a sample or stream.

Values

MIDAS_LOOP_NO Sample or stream does not loop

MIDAS_LOOP_YES Sample or stream loops

4.1.3 MIDASpanning

```
enum MIDASpanning
```

Description

These constants are used to describe the panning position of a sound. Legal panning positions range from -64 (extreme left) to 64 (extreme right), inclusive, plus MIDAS_PAN_SURROUND for surround sound.

Values

MIDAS_PAN_LEFT Panning position full left

MIDAS_PAN_MIDDLE Panning position middle

MIDAS_PAN_RIGHT Panning position full right

MIDAS_PAN_SURROUND Surround sound

4.1.4 MIDASchannels

```
enum MIDASchannels
```

Description

These constants are used to indicate the channel number a sound should be played on. Legal channel numbers range from 0 upwards, depending on the number of open channels. In addition, MIDAS_CHANNEL_AUTO can be used with *MIDASplaySample*.

Values

MIDAS_CHANNEL_AUTO Select channel automatically, used with *MIDASplaySample*

4.2 Data types

This section describes all data types used in MIDAS sample playback.

4.2.1 MIDASsample

```
typedef ... MIDASsample;
```

Description

MIDASsample is a sample handle that defines a sample that has been loaded into memory. The sample handle is used for identifying the sample when playing or deallocating it.

4.2.2 MIDASsamplePlayHandle

```
typedef ... MIDASsamplePlayHandle;
```

Description

MIDASsamplePlayHandle is a sample playing handle. It describes a sample sound that is being played. The sample playing handle is used for controlling the attributes of the sound, such as panning or volume, and for stopping the sound.

4.3 Functions

This section describes all functions available for MIDAS sample playback.

4.3.1 MIDASloadRawSample

```
MIDASsample MIDASloadRawSample(char *filename, int sampleType,
    int loopSample)
```

Loads a raw sound effect sample.

Input

filename Sample file name

sampleType Sample type, see `enum` *MIDASsampleTypes*

loopSample Sample loop type, see `enum` *MIDASloop*

Description

This function loads a sound effect sample into memory and adds it to the Sound Device. The sample file must contain just the raw sample data, and all of it will be loaded into memory. If **loopSample** is MIDAS_LOOP_YES, the whole sample will be looped. After the sample has been loaded, it can be played using *MIDASplaySample*, and it should be deallocated with *MIDASfreeSample* after it is no longer used.

Return value

Sample handle if successful, NULL if failed.

Operating systems

All

See also

MIDASplaySample, *MIDASfreeSample*

4.3.2 MIDASfreeSample

```
BOOL MIDASfreeSample(MIDASsample sample)
```

Deallocates a sound effect sample.

Input

sample Sample to be deallocated

Description

This function deallocates a sound effect sample that has been previously loaded with *MIDASloadRawSample*. It removes the sample from the Sound Device and deallocates the memory used. This function may not be called if the sample is still being played.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASloadRawSample

4.3.3 MIDASsetAutoEffectChannels

```
BOOL MIDASsetAutoEffectChannels(unsigned firstChannel,
    unsigned numChannels)
```

Sets the range of channels that can be used as automatic effect channels.

Input

firstChannel First channel that can be used

numChannels Number of channels that can be used

Description

This function is used to set the range of channels that can be used as automatic effect channels by *MIDASplaySample*. When *MIDASplaySample* is passed MIDAS_CHANNEL_AUTO as the channel number, it will use one of these automatic channels to play the sound.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplaySample

4.3.4 MIDASplaySample

```
MIDASsamplePlayHandle MIDASplaySample(MIDASsample sample,
    unsigned channel, int priority, unsigned rate,
    unsigned volume, int panning)
```

Plays a sound effect sample.

Input

sample The sample that will be played

channel The channel number that is used to play the sample. Use MIDAS_CHANNEL_AUTO to let *MIDASplaySample* select the channel automatically. See `enum` *MIDASchannels*.

priority Sample playing priority. The higher the value the more important the sample is considered.

rate Initial sample rate for the sample

volume Initial volume for the sample

panning Initial panning position for the sample. See `enum` *MIDASpanning*.

Description

This function is used to play a sound effect sample on a given channel. The sample will receive as initial parameters the values passed as arguments, and playing the sample will be started. If **channel** is MIDAS_CHANNEL_AUTO, the channel will be selected automatically. The sample playing priority is used to choose the channel the sample will be played on in this case.

This function returns a sample playing handle, that can later be used to stop the sample or change its parameters. This makes it possible to refer to samples without knowing the exact channel they are played on.

Return value

Sample playing handle if successful, NULL if failed.

Operating systems

All

See also

MIDASstopSample, *MIDASsetAutoEffectChannels*

4.3.5 MIDASstopSample

```
BOOL MIDASstopSample(MIDASsamplePlayHandle sample)
```

Stops playing a sample.

Input

sample Sample to be stopped

Description

This function stops playing a sound effect sample started with *MIDASplaySample*. Playing the sound will stop, and the channel is freed for other samples to use. Note that **sample** is the sample playing handle returned by *MIDASplaySample*.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplaySample

4.3.6 MIDASsetSampleRate

```
BOOL MIDASsetSampleRate(MIDASsamplePlayHandle sample,
    unsigned rate)
```

Changes the sample rate for a sound effect sample.

Input

sample Sample to be changed

rate New sample rate for the sample

Description

This function changes the sample rate for a sound effect sample that is being played.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplaySample

4.3.7 MIDASsetSampleVolume

```
BOOL MIDASsetSampleVolume(MIDASsamplePlayHandle sample,
    unsigned volume)
```

Changes the volume for a sound effect sample.

Input

sample Sample to be changed

rate New volume for the sample (0–64)

Description

This function changes the volume for a sound effect sample that is being played.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplaySample

4.3.8 MIDASsetSamplePanning

```
BOOL MIDASsetSamplePanning(MIDASsamplePlayHandle sample,
    int panning)
```

Changes the panning position of a sound effect sample.

Input

sample Sample to be changed

panning New panning position for the sample (see `enum` *MIDASpanning*)

Description

This function changes the panning position of a sound effect sample that is being played. See description of `enum` *MIDASpanning* for information about the panning position values.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplaySample

4.3.9 MIDASsetSamplePriority

```
BOOL MIDASsetSamplePriority(MIDASsamplePlayHandle sample,
    int priority)
```

Changes the playing priority of a sound effect sample.

Input

sample Sample to be changed

priority New playing priority for the sample

Description

This function changes the playing priority a sound effect sample that is being played.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplaySample

Chapter 5

Stream playback

5.1 Constants

This section describes all constants used in MIDAS stream playback. They are grouped according to the `enum` used to define them. Note that stream playback properties, such as volume and panning, are controlled similarily those of samples.

5.2 Data types

This section describes all data types used in MIDAS stream playback.

5.2.1 MIDASstreamHandle

```
typedef ... MIDASstreamHandle;
```

Description

MIDASstreamHandle is a stream handle that defines a digital audio stream that is being played. Streams only exist in the system when they are being played, so there is no separate "playing handle" data type.

5.3 Functions

This section describes all functions available for MIDAS stream playback.

5.3.1 MIDASplayStreamFile

```
MIDASstreamHandle MIDASplayStreamFile(unsigned channel,
    char *fileName, unsigned sampleType, unsigned sampleRate,
    unsigned bufferLength, int loopStream)
```

Starts playing a digital audio stream from a file.

Input

channel The channel number the stream will be played on

fileName Stream file name

sampleType Stream sample type, see `enum` *MIDASsampleTypes*

sampleRate Stream sample rate

bufferLength Stream playback buffer length in milliseconds

loopStream 1 if the stream should be looped, 0 if not

Description

This function starts playing a digital audio stream from a file. It allocates the stream buffer, creates a new thread that will read sample data from the file to the stream buffer, and starts the Sound Device to play the stream. The stream will continue playing until it is stopped with *MIDASstopStream*. When a stream is being played on a channel, that channel may not be used for other purposes.

The stream buffer length should be at least around 500ms if the stream file is being read from a disc, to avoid breaks in stream playback

Return value

MIDAS stream handle if successful, NULL if failed.

Operating systems

Win32, Linux

See also

MIDASstopStream

5.3.2 MIDASstopStream

```
BOOL MIDASstopStream(MIDASstreamHandle stream)
```

Stops playing a digital audio stream.

Input

stream The stream that will be stopped

Description

This function stops playing a digital audio stream. It stops the stream player thread, deallocates the stream buffer and closes the stream file. The stream playback channel can then be used for other purposes.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32, Linux

See also

MIDASplayStream

5.3.3 MIDASplayStreamPolling

```
MIDASstreamHandle MIDASplayStreamPolling(unsigned channel,
    unsigned sampleType, unsigned sampleRate,
    unsigned bufferLength)
```

Starts playing a digital audio stream in polling mode.

Input

channel The channel number the stream will be played on

sampleType Stream sample type, see `enum` *MIDASsampleTypes*

sampleRate Stream sample rate

bufferLength Stream playback buffer length in milliseconds

Description

This function starts playing a digital audio stream in polling mode. It allocates and empty stream buffer, and starts the Sound Device to play the stream. Sample data can be fed to the stream buffer with *MIDASfeedSteramData*. The stream will continue playing until it is stopped with *MIDASstopStream*. When a stream is being played on a channel, that channel may not be used for other purposes.

To avoid breaks in playback, the stream buffer size should be at least twice the expected polling period. That is, if you will be feeding data 5 times per second (every 200ms), the buffer should be at least 400ms long.

Return value

MIDAS stream handle if successful, NULL if failed.

Operating systems

Win32, Linux

See also

MIDASstopStream, *MIDASfeedStreamData*

5.3.4 MIDASfeedStreamData

```
unsigned MIDASfeedStreamData(MIDASstreamHandle stream,
    unsigned char *data, unsigned numBytes, BOOL feedAll);
```

Feeds sound data to a digital audio stream buffer.

Input

stream The stream that will play the data

data Pointer to sound data

numBytes Number of bytes of sound data available

feedAll TRUE if the function should block until all sound data can be fed

Description

This function is used to feed sample data to a stream that has been started with *MIDASplayStreamPolling*. Up to **numBytes** bytes of new sample data from ***data** will be copied to the stream buffer, and the stream buffer write position is updated accordingly. The function returns the number of bytes of sound data actually used. If **feedAll** is TRUE, the function will block the current thread of execution until all sound data is used.

Return value

The number of bytes of sound data actually used.

Operating systems

Win32, Linux

See also

MIDASplayStreamPolling

5.3.5 MIDASsetStreamRate

```
BOOL MIDASsetStreamRate(MIDASstreamHandle stream,
    unsigned rate);
```

Changes stream playback sample rate.

Input

stream Stream handle for the stream

rate New playback sample rate for the stream, in Hertz.

Description

This function changes the playback sample rate for a stream that is being played. The initial sample rate is given as an argument to the function that starts stream playback.

Note that the stream playback buffer size is calculated based on the initial sample rate, so the stream sample rate should not be changed very far from that figure. In particular, playback sample rates over two times the initial value may cause breaks in stream playback. Too low rates, on the other hand, will increase latency.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32, Linux

See also

MIDASsetStreamVolume, *MIDASsetStreamPanning*

5.3.6 MIDASsetStreamVolume

```
BOOL MIDASsetStreamVolume(MIDASstreamHandle stream,
    unsigned volume);
```

Changes stream playback volume.

Input

stream Stream handle for the stream

volume New volume for the stream, 0–64.

Description

This function changes the playback volume for a stream that is being played. The initial volume is 64 (maximum).

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32, Linux

See also

MIDASsetStreamRate, *MIDASsetStreamPanning*

5.3.7 MIDASsetStreamPanning

```
BOOL MIDASsetStreamPanning(MIDASstreamHandle stream,
    int panning);
```

Changes stream panning position.

Input

stream Stream handle for the stream

panning New panning position for the stream

Description

This function changes the panning position for a stream that is being played. The initial volume is 0 (center). See description of `enum` *MIDASpanning* for information about the panning position values.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32, Linux

See also

MIDASsetStreamVolume, *MIDASsetStreamRate*