MIDAS Digital Audio System

API Reference

Petteri Kangaslampi

March 1, 1998

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 seven different chapters, according to different functional groups. In addition to this introduction, the chapters cover configuration and initialization, overall system control, module playback, sample playback, stream playback, and miscellaneous system functions. 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 MIDASdsoundModes

```
enum MIDASdsoundModes
```

Description

These constants are used to describe different MIDAS DirectSound usage modes. By default MIDAS does not use DirectSound at all, and DirectSound usage can be enabled by setting *MIDAS_OPTION_DSOUND_MODE*. Note that *MIDAS_OPTION_DSOUND_HWND* needs to be set when using DirectSound. A complete discussion of using DirectSound with MIDAS is available at MIDAS Programmer's Guide.

Values

MIDAS_DSOUND_DISABLED DirectSound usage is disabled

MIDAS_DSOUND_STREAM DirectSound is used in stream mode – MIDAS will play to a DirectSound stream buffer. DirectSound usage is disabled if DirectSound runs in emulation mode.

MIDAS_DSOUND_PRIMARY DirectSound is used in primary buffer mode if possible – MIDAS will play directly to DirectSound primary buffer. If primary buffer is not available for writing, this mode behaves like MIDAS_DSOUND_STREAM.

MIDAS_DSOUND_FORCE_STREAM Behaves like MIDAS_DSOUND_STREAM, except that DirectSound is used always, even in emulation mode.

2.1.2 MIDASfilterModes

```
enum MIDASfilterModes
```

Description

These constants are used to describe different MIDAS output filter modes. By default, MIDAS Digital Audio System will select the most appropriate filter for the mixing mode automatically, but in some cases manually overriding the selection can result in better sound. Also, if all sounds are played exactly at the mixing rate, the output filtering should be disabled to get better sound quality. Finally, filtering is not normally used with high-quality mixing, and is therefore disabled with it by default.

The filter mode can be set by changing the option *MIDAS_OPTION_FILTER_MODE* with the function *MIDASsetOption*.

Values

MIDAS_FILTER_NONE No filtering

MIDAS_FILTER_LESS Some filtering

MIDAS_FILTER_MORE More filtering

MIDAS_FILTER_AUTO Automatic filter selection (default)

2.1.3 MIDASmixingModes

```
enum MIDASmixingModes
```

Description

These constants define the different MIDAS Digital Audio System mixing modes available. The high-quality mixing mode yields better sound quality, with virtually no aliasing noise, but uses much more CPU power than normal mixing, and is mainly intended for stand-alone module players and similar applications. By default, MIDAS uses the normal mixing quality.

The mixing mode can be set by changing the option *MIDAS_OPTION_MIXING_MODE* with the function *MIDASsetOption*.

Values

MIDAS_MIX_NORMAL_QUALITY Normal quality mixing

MIDAS_MIX_HIGH_QUALITY High-quality interpolating mixing

2.1.4 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

2.1.5 MIDASoptions

`enum MIDASoptions`

Description

These constants are used with the function *MIDASsetOption* to change different MIDAS configuration options, and *MIDASgetOption* to query their current settings.

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

MIDAS_OPTION_DSOUND_MODE DirectSound mode to use, see `enum` *MIDASdsoundModes*

MIDAS_OPTION_DSOUND_HWND Window handle for DirectSound support. The window handle is used by DirectSound to determine which window has the focus. The window handle has to be set when using DirectSound.

MIDAS_OPTION_DSOUND_OBJECT The DirectSound object that should be used. Setting this option forces DirectSound support on.

MIDAS_OPTION_DSOUND_BUFLEN Output buffer length fot DirectSound, in milliseconds. This option is used instead of MIDAS_OPTION_MIXBUFLEN when using DirectSound without emulation.

MIDAS_OPTION_16BIT_ULAW_AUTOCONVERT Controls whether 16-bit samples will be automatically converted to u-law or not. Enabled by default. The autoconversion only applies to Sound Devices which can natively play u-law format data, and will only be used if it results in smaller CPU use.

MIDAS_OPTION_FILTER_MODE Output filter mode, see `enum` *MIDASfilterModes*. The filter is selected automatically by default.

MIDAS_OPTION_MIXING_MODE Mixing mode, affects the output sound quality. See `enum` *MIDASmixingModes*.

MIDAS_OPTION_DEFAULT_STEREO_SEPARATION Controls the default stereo separation for modules with no panning information (MODs and old S3Ms). 64 is maximum stereo separation, 0 none. Default 64.

MIDAS_OPTION_FORCE_NO_SOUND Forces the No Sound Sound Device to be used for playback. Useful for trying to run MIDAS with no sound if *MIDASinit* fails.

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 and configuration.

2.3.1 MIDASclose

```
BOOL MIDASclose(void)
```

Uninitializes MIDAS Digital Audio System.

Input

None.

Description

This function uninitializes all MIDAS Digital Audio 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

2.3.2 MIDASconfig

```
BOOL MIDASconfig(void)
```

Runs manual MIDAS setup.

Input

None.

Description

This function runs the manual MIDAS Digital Audio System configuration. It queries the sound card to use and desired sound quality and output mode from the user. The setup entered can be saved do disk with *MIDASsaveConfig* and loaded back with *MIDASloadConfig*, or written to registry with *MIDASwriteConfigRegistry* and read back with *MIDASreadConfigRegistry*. These functions can be used to create a simple external setup program, or just save the settings between two runs of the program. After this function has been called, *MIDASsetOption* can be used to change the output mode options, to, for example, force mono output.

This function returns TRUE if the setup was completed successfully, FALSE if not. The setup can fail for two reasons: either the user aborted it by pressing escape or clicking Cancel, or an error occured. As errors during the setup are extremely unlikely, it is safe to simply exit the program if this function returns FALSE. *MIDASgetLastError* can be used to check if an error occured — if the return value is zero, the user just pressed cancelled the setup.

This function must be called before *MIDASinit*, but after *MIDASstartup*.

Return value

TRUE if successful, FALSE if not (the user cancelled the configuration, or an error occurred)

Operating systems

MS-DOS, Win32

See also

MIDASsaveConfig, *MIDASloadConfig*, *MIDASsetOption*, *MIDASgetOption*, *MIDASinit*, *MIDASwriteConfigRegistry*, *MIDASreadConfigRegistry*

2.3.3 MIDASdetectSoundCard

```
BOOL MIDASdetectSoundCard(void)
```

Attempts to detect the sound card to use.

Input

None.

Description

[MS-DOS only]

This function attempts to detect the sound card that should be used. It will set up MIDAS to use the detected card, and return TRUE if a sound card was found, FALSE if not. If this function returns FALSE, you should run *MIDASconfig* to let the user manually select the sound card. Note that you **can** use MIDAS even if no sound card has been selected - MIDAS will just not play sound in that case.

If no sound card has been manually set up, *MIDASinit* will automatically detect it, or use No Sound if none is available. Therefore this function does not have to be called if manual setup will not be used.

Note that, as there is no way to safely autodetect the Windows Sound System cards under MS-DOS, MIDAS will not attempt to detect them at all. If you do not provide a manual setup possibility to your program (via *MIDASconfig*), WSS users will not be able to get any sound. The computer may also have several sound cards, and the user may wish not to use the one automatically detected by MIDAS. Therefore it is a very good idea to include an optional manual sound setup to all programs.

This discussion naturally applies to MS-DOS only, under Win32 and Linux MIDAS uses the sound card through the system audio devices, and no sound card selection or setup is necessary.

Return value

TRUE if a sound card was detected, FALSE if not.

Operating systems

MS-DOS

See also

MIDASconfig, MIDASinit

2.3.4 MIDASgetDisplayRefreshRate

```
DWORD MIDASgetDisplayRefreshRate(void)
```

Gets the current display refresh rate.

Input

None.

Description

This function tries to determine the current display refresh rate. It is used with *MIDASsetTimer-Callbacks* to set a display-synchronized timer callback. It returns the current display refresh rate in milliHertz (ie. 1000*Hz, 50Hz becomes 50000, 70Hz 70000 etc), or 0 if it could not determine the refresh rate. The refresh rate may be unavailable when running under Win95 or a similar OS, or when the VGA card does return Vertical Retraces correctly (as some SVGA cards do in SVGA modes). Therefore it is important to check the return value, and substitute some default value if it is zero.

Unlike most other MIDAS functions, this function must be called **before** *MIDASinit* is called, as the MIDAS timer may interfere with the measurements.

Note that the display refresh rate is **display mode specific**. Therefore you need to set up the display mode with which you want to use display-synchronized timer callbacks **before** calling this function. Also, if your application uses several display modes, you must get the display refresh rate for each mode separately, and remove and restart the display-synchronized timer callbacks at each mode change.

This function is only available in MS-DOS.

Return value

The current display refresh rate, in milliHertz, or 0 if unavailable.

Operating systems

MS-DOS

See also

MIDASsetTimerCallbacks

2.3.5 MIDASgetOption

```
DWORD MIDASgetOption(int option)
```

Gets a MIDAS option.

Input

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

Description

This function reads the current value of a MIDAS option. The different number codes for different options are described above.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASsetOption

2.3.6 MIDASinit

```
BOOL MIDASinit(void)
```

Initializes MIDAS Digital Audio System.

Input

None.

Description

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

If this function fails with no apparent reason, it is very probable that some other application is using the sound card hardware or the user has badly mis-configured MIDAS. Therefore, if this function fails, it is recommended to give the user a possibility to close other applications and retry, continue with no sound (set option *MIDAS_OPTION_FORCE_NO_SOUND*), or possibly re-configure MIDAS.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASsetOption, *MIDASclose*

2.3.7 MIDASloadConfig

```
BOOL MIDASloadConfig(char *fileName)
```

Load MIDAS setup from disk.

Input

fileName Setup file name

Description

This function loads MIDAS setup from disk. The setup must have been saved with *MIDASsaveConfig*. *MIDASsetOption* can be used afterwards to change, for example, the output mode.

This function must be called before *MIDASinit*, but after *MIDASstartup*.

Return value

TRUE if successful, FALSE if not.

Operating systems

MS-DOS, Win32

See also

MIDASconfig, *MIDASsaveConfig*

2.3.8 MIDASreadConfigRegistry

```
BOOL MIDASreadConfigRegistry(DWORD key, char *subKey);
```

Reads MIDAS configuration from a registry key.

Input

key A currently open registry key, for example **H_KEY_CURRENT_USER**

subKey The name of the subkey of **key** that contains the configuration

Description

This function reads the MIDAS configuration from a registry key. The configuration must have been written there with *MIDASwriteConfigRegistry*. *MIDASsetOption* can be used afterwards to chance, for example, the output mode, and *MIDASconfig* to prompt the user for new settings.

This function must be called before *MIDASinit*, but after *MIDASstartup*.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32

See also

MIDASconfig, *MIDASwriteConfigRegistry*

2.3.9 MIDASsaveConfig

```
BOOL MIDASsaveConfig(char *fileName)
```

Saves the MIDAS setup to a file.

Input

fileName Setup file name

Description

This function saves the MIDAS setup entered with *MIDASconfig* to a file on disk. It can be then loaded with *MIDASloadConfig*.

Return value

TRUE if successful, FALSE if not.

Operating systems

MS-DOS, Win32

See also

MIDASconfig, *MIDASloadConfig*

2.3.10 MIDASsetOption

```
BOOL MIDASsetOption(int option, DWORD 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

2.3.11 MIDASstartup

```
BOOL MIDASstartup(void)
```

Prepares MIDAS Digital Audio System for initialization and use.

Input

None.

Description

This function sets all MIDAS Digital Audio System 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*, or call *MIDASconfig* to prompt the user for configuration options.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASsetOption, *MIDASinit*, *MIDASclose*

2.3.12 MIDASwriteConfigRegistry

```
BOOL MIDASwriteConfigRegistry(DWORD key, char *subKey);
```

Writes the MIDAS configuration to a registry key.

Input

key A currently open registry key, for example **H_KEY_CURRENT_USER**

subKey The name of the subkey of **key** where the configuration will be written. They key does not need to exist.

Description

This function saves the current MIDAS configuration, usually entered with *MIDASconfig* to a registry key. The configuration can then be read with *MIDASreadConfigRegistry*.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32

See also

MIDASconfig, *MIDASreadConfigRegistry*

Chapter 3

System control

3.1 Constants

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

3.2 Data types

This section describes all data types used in MIDAS system control.

3.3 Functions

This section describes all functions available for MIDAS system control. This includes error handling.

3.3.1 MIDASallocateChannel

```
DWORD MIDASallocateChannel(void)
```

Allocates a single Sound Device channel.

Input

None.

Description

This function allocates a single Sound Device channel, and returns its number. Sound Device are used for all sound playback, but most functions take care of allocating and deallocating channels automatically. If you wish to play a sample on a specific channel, to ensure it won't be replaced by other samples, you'll need to pass *MIDASplaySample* a specific channel number, and this function is used to allocate those channels.

Channels allocated with this function need to be deallocated with *MIDASfreeChannel*. Before any channels can be allocated, some sound channels need to be opened with *MIDASopenChannels*.

Return value

Channel number for the allocated channel, or MIDAS_ILLEGAL_CHANNEL if failed.

Operating systems

All

See also

MIDASopenChannels, MIDASfreeChannel.

3.3.2 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*

3.3.3 MIDASfreeChannel

```
BOOL MIDASfreeChannel(DWORD channel)
```

Deallocates a single Sound Device channel.

Input

channel The channel number to deallocate, from *MIDASallocateChannel*

Description

This function deallocates a single Sound Device channel that has previously been allocated with *MIDASallocateChannel*. Any sound playback on the channel should be stopped before deallocating it.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASallocateChannel

3.3.4 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

3.3.5 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

3.3.6 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 Digital Audio System version number as a string.

Operating systems

Win32, Linux

See also

3.3.7 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.

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 available channels 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

3.3.8 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*

3.3.9 MIDASremoveTimerCallbacks

```
BOOL MIDASremoveTimerCallbacks(void)
```

Removes the user timer callbacks.

Input

None.

Description

This function removes the user timer callbacks set with *MIDASsetTimerCallbacks*. The callback functions will no longer be called. This function **may not** be called if *MIDASsetTimerCallbacks* has not been called before.

It is not necessary to call this function without exiting even if the callbacks have been used — *MIDASclose* will remove the callbacks automatically. On the other hand, if the callback functions or rate are changed with *MIDASsetTimerCallbacks*, this function must be called to remove the previous ones first.

Return value

TRUE if successful, FALSE if not.

Operating systems

MS-DOS

See also

MIDASsetTimerCallbacks

3.3.10 MIDASresume

```
BOOL MIDASresume(void)
```

Resumes MIDAS sound playback.

Input

None.

Description

This function re-allocates the system audio device to MIDAS, and resumes sound playback, after being suspended with *MIDASsuspend*. See *MIDASsuspend* documentation for more information about suspending MIDAS.

Note that this function may fail, if another application has captured the sound output device while MIDAS was suspended.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32

See also

MIDASsuspend

3.3.11 MIDASsetAmplification

```
BOOL MIDASsetAmplification(DWORD amplification)
```

Sets sound output amplification level.

Input

amplification New output amplification level

Description

This function changes the output amplification level. Amplification can be used to boost the volume of the music, if the sounds played are unusually quiet, or lower it if the output seems distorted.. The amplification level is given as a percentage — 100 stands for no amplification, 200 for double volume, 400 for quadruple volue, 50 for half volume etc.

MIDAS has some build-in amplification, but the default amplification is designed for situations where most channels have data played at moderate volumes (eg. module playback). If a lot of the channels are empty, or sounds are played at low volumes, adding amplification with this function can help to get the total sound output at a reasonable level. The amplification set with this function acts on top of the default MIDAS amplification, so nothing will be overridden.

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

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

3.3.12 MIDASsetTimerCallbacks

```
BOOL MIDASsetTimerCallbacks(DWORD rate, BOOL displaySync,
    void (MIDAS_CALL *preVR)(), void (MIDAS_CALL *immVR)(),
    void (MIDAS_CALL *inVR)());
```

Sets the user timer callback functions and their rate.

Input

rate Timer callback rate, in milliHertz (1000*Hz, 100Hz becomes 100000 etc)

displaySync TRUE if the callbacks should be synchronized to display refresh, FALSE if not.

preVR preVR callback function pointer or NULL

immVR immVR callback function pointer or NULL

inVR inVR callback function pointer or NULL

Description

This function sets the user timer callback functions and their call rate. The functions will be called periodically by the MIDAS timer interrupt, one after another. Any of the callback function pointers may be set to NULL — the callback is then ignored.

If **displaySync** is TRUE, the timer system attempts to synchronize the callbacks to the display refresh. In that case, **preVR** is called just before the Vertical Retrace starts, **immVR** immediately after it has started, and **inVR** later during retrace. **preVR** can then be used for changing the display start address, for example. If display synchronization is used, **rate** has to be set to the value returned by *MIDASgetDisplayRefreshRate*.

If **displaySync** is FALSE, or the timer system is unable to synchronize to display refresh (running under Win95, for example), the functions are simply called one after another: first **preVR**, then **immVR** and last **inVR**. Note that display synchronization is not always possible, and this may happen even if **displaySync** is set to 1.

In either case, both the **preVR** and **immVR** functions have to be kept as short as possible, to prevent timing problems. They should not do more than update a few counters, or set a couple of hardware registers. **inVR** can take somewhat longer time, and be used for, for example, setting the VGA palette. It should not take more than one quarter of the time between callbacks though.

The most common use for the timer callback functions is to use them for controlling the program speed. There one of the callbacks, usually **preVR** is simply used for incrementing a counter. This counter then can be used to determine when to display a new frame of graphics, for example, or how many frames of movement needs to be skipped to maintain correct speed.

Note that this function may cause a small break to music playback with some sound cards. Therefore it should not be called more often than necessary. Also, if the application changes display modes, any display-synchronized timer callbacks **must** be resetted, and a separate refresh rate must be read for each display mode used.

`MIDAS_CALL` is the calling convention used for the callback functions — `__cdecl` for Watcom C, empty (default) for DJGPP. As the functions will be called from an interrupt, the module containing the callback functions must be compiled with the "SS==DS" setting disabled (command line argument "-zu" for Watcom C, default setting for DJGPP).

Return value

TRUE if successful, FALSE if not.

Operating systems

MS-DOS

See also

MIDASremoveTimerCallbacks, MIDASgetDisplayRefreshRate

3.3.13 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*

3.3.14 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

3.3.15 MIDASsuspend

```
BOOL MIDASsuspend(void)
```

Suspends MIDAS Digital Audio System.

Input

None.

Description

This function suspends all MIDAS Digital Audio System output, and releases the system audio device to other programs. Playback can be resumed with *MIDASresume*. Suspending and resuming MIDAS can be used to change some of the initial configuration options (set with *MIDASsetOption*) on the fly. In particular, the DirectSound mode and DirectSound window handle can be changed while MIDAS is suspended, and the new values take effect when MDIAS is restarted. Buffer size can also be changed, although this is not recommended. Output mode and mixing rate cannot be changed without completely uninitializing MIDAS.

While MIDAS is suspended, all MIDAS functions can be called normally — the sound simply is not played. Also, stream, module and sample playback positions do not change while MIDAS is suspended.

Note that *MIDASsuspend* and *MIDASresume* are only available in Win32 systems at the moment.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32

See also

MIDASresume

Chapter 4

Module playback

4.1 Constants

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

4.2 Data types

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

4.2.1 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.

4.2.2 MIDASmodule

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

Description

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

4.2.3 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.

4.2.4 MIDASmodulePlayHandle

```
typedef ... MIDASmodulePlayHandle;
```

Description

MIDASmodulePlayHandle is a module playback handle that defines a module or module section that is being played. One module can be played several times simultaneously.

4.2.5 MIDASplayStatus

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

songLoopCount Module song loop counter — incremented by 1 every time the song loops around. Module looping can be detected by checking if this field is nonzero.

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 FastTracker 2 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.

4.3 Functions

This section describes all functions available for MIDAS module playback.

4.3.1 MIDASfadeMusicChannel

```
BOOL MIDASfadeMusicChannel(MIDASmodulePlayHandle playHandle,
    unsigned channel, unsigned fade)
```

Fades a music channel.

Input

playHandle Module playback handle for the music

channel Module channel number to control

fade Channel fade value: 64 is normal volume, 0 silence

Description

This function is used to control the “fade” of a music channel. Channel fade acts as a channel-specific master volume: it can be used to quiet down the general volume of the sounds played on the channel, while any volume changes in the music still take effect.

channel if the **module** channel number for the sounds to control, not a Sound Device channel number. Module channels are numbered from zero upwards.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASsetMusicVolume

4.3.2 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

4.3.3 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

4.3.4 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 will 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*

4.3.5 MIDASgetPlayStatus

```
BOOL MIDASgetPlayStatus(MIDASmodulePlayHandle playHandle,
    MIDASplayStatus *status)
```

Gets module playback status.

Input

playHandle Module playback handle for the music

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

4.3.6 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*

4.3.7 MIDASplayModule

```
MIDASmodulePlayHandle MIDASplayModule(MIDASmodule module,
    BOOL loopSong)
```

Starts playing a module.

Input

module Module to be played

loopSong TRUE if the song should be looped, FALSE if not

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. This function plays the complete module — to play just a section of the song data, use *MIDASplayModuleSection*.

Note! Currently, when multiple modules or module sections are played simultaneously, all modules should have the same (BPM) **tempo**. Otherwise some modules may be played at the wrong tempo. All modules can have different **speed** setting though.

Return value

MIDAS module playback handle for the module, or 0 if failed.

Operating systems

All

See also

MIDASloadModule, *MIDASstopModule*, *MIDASplayModuleSection*

4.3.8 MIDASplayModuleSection

```
MIDASmodulePlayHandle MIDASplayModuleSection(MIDASmodule module,
    unsigned startPos, unsigned endPos, unsigned restartPos,
    BOOL loopSong)
```

Starts playing a module section.

Input

module Module to be played

startPos Start song position for the section to play

endPos End song position for the section to play

restartPos Restart position to use when the section loops

loopSong TRUE if the playback should be looped, FALSE if not

Description

This module starts playing a section of 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. Playback will start from the pattern at position **startPos**, and after the pattern at position **endPos** has been played, playback will resume from the pattern at **restartPos**. This function can thus be used to play a section of a module, and a single module can be used to store several songs.

Note! Currently, when multiple modules or module sections are played simultaneously, all modules should have the same (BPM) **tempo**. Otherwise some modules may be played at the wrong tempo. All modules can have different **speed** setting though.

Return value

MIDAS module playback handle for the module, or 0 if failed.

Operating systems

All

See also

MIDASloadModule, *MIDASstopModule*, *MIDASplayModule*

4.3.9 MIDASsetMusicSyncCallback

```
BOOL MIDASsetMusicSyncCallback(MIDASmodulePlayHandle playHandle,
    void (MIDAS_CALL *callback)
    (unsigned syncInfo, unsigned position, unsigned row))
```

Sets the music synchronization callback.

Input

playHandle Module playback handle for the music

callback Pointer to the callback function, NULL to disable

Description

This function sets the music synchronization callback function. It will be called by the MIDAS music player each time a **Wxx** command is played from a FastTracker 2 or Scream Tracker 3 module, or a **Zxx** command from an Impulse Tracker module. The function will receive as its arguments the synchronization command infobyte (xx), the current playback position and the current playback row. Setting **callback** to NULL disables it.

`MIDAS_CALL` is the calling convention used for the callback function — `__cdecl` for Watcom and Visual C/C++, empty (default) for GCC. Under MS-DOS the function will be called from the MIDAS timer interrupt, so the module containing the callback function must be compiled with the "SS==DS" setting disabled (command line argument "-zu" for Watcom C, default setting for DJGPP). Under Win32 and Linux the function will be called in the context of the MIDAS player thread.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

4.3.10 MIDASsetMusicVolume

```
BOOL MIDASsetMusicVolume(MIDASmodulePlayHandle playHandle,
    unsigned volume)
```

Changes music playback volume.

Input

playHandle Module playback handle for the music

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

4.3.11 MIDASsetPosition

```
BOOL MIDASsetPosition(MIDASmodulePlayHandle playHandle,
    int newPosition)
```

Changes module playback position.

Input

playHandle Module playback handle for the music

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*

4.3.12 MIDASstopModule

```
BOOL MIDASstopModule(MIDASmodulePlayHandle playHandle)
```

Stops playing a module.

Input

playHandle Module playback handle for the music that should be stopped.

Description

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

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASplayModule, *MIDASopenChannels*

Chapter 5

Sample playback

5.1 Constants

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

5.1.1 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*

MIDAS_ILLEGAL_CHANNEL Illegal channel number, returned by *MIDASallocateChannel* as an error code.

5.1.2 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

5.1.3 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

5.1.4 MIDASsamplePlayStatus

```
enum MIDASsamplePlayStatus
```

Description

These constants describe the possible sample playing status values returned by *MIDASgetSamplePlayStatus*.

Values

MIDAS_SAMPLE_STOPPED The sample has stopped playing. Either the sample has ended, or another sample has taken its place (with automatic effect channels)

MIDAS_SAMPLE_PLAYING The sample is playing

5.1.5 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

MIDAS_SAMPLE_ADPCM_MONO 4-bit ADPCM mono sample (streams only)

MIDAS_SAMPLE_ADPCM_STEREO 4-bit ADPCM stereo sample (streams only)

MIDAS_SAMPLE_ULAW_MONO 8-bit *mu*-law mono sample

MIDAS_SAMPLE_ULAW_STEREO 8-bit *mu*-law stereo sample

5.2 Data types

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

5.2.1 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.

5.2.2 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.

5.3 Functions

This section describes all functions available for MIDAS sample playback.

5.3.1 MIDASallocAutoEffectChannels

```
BOOL MIDASallocAutoEffectChannels(unsigned numChannels)
```

Allocates a number of channels for use as automatic effect channels.

Input

numChannels Number of channels to use

Description

This function allocates a number 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. The channels allocated can be deallocated with *MIDASfreeAutoEffectChannels*.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASfreeAutoEffectChannels, MIDASplaySample

5.3.2 MIDASfreeAutoEffectChannels

```
BOOL MIDASfreeAutoEffectChannels(void)
```

Deallocates the channels allocated for automatic effect channels.

Input

None.

Description

This function deallocates the channels allocated by *MIDASallocAutoEffectChannels* for use as automatic sound effect channels.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASallocAutoEffectChannels

5.3.3 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* or *MIDASloadWaveSample*. 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, *MIDASloadWaveSample*

5.3.4 MIDASgetSamplePlayStatus

```
DWORD MIDASgetSamplePlayStatus(MIDASsamplePlayHandle sample);
```

Gets the sample playing status.

Input

sample The sample playing handle

Description

This function returns the playing handle for the sample with playing handle **sample**. The playing status can be used to determine if the sample has stopped, and if another sample should be played in its place. Normally manually checking for the sample playing status is unnecessary, however, as new samples can simply be played on top of the old ones.

Return value

Playing status for the sample, see `enum` *MIDASsamplePlayStatus*.

Operating systems

All

See also

MIDASplaySample, MIDASstopSample

5.3.5 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

5.3.6 MIDASloadWaveSample

```
MIDASsample MIDASloadWaveSample(char *filename, int loopSample)
```

Loads a RIFF WAVE sound effect sample.

Input

filename Sample file name

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 be a standard RIFF WAVE (`.wav`) sound file, containing raw PCM sound data — compressed WAVE files are not supported. 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

5.3.7 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. Otherwise a channel needs to be allocated with *MIDASallocateChannel* before the sample can be played.

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, MIDASallocAutoEffectChannels

5.3.8 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

5.3.9 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

5.3.10 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

5.3.11 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

5.3.12 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

Chapter 6

Stream playback

6.1 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.

6.2 Data types

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

6.2.1 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.

6.3 Functions

This section describes all functions available for MIDAS stream playback.

6.3.1 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

6.3.2 MIDASgetStreamBytesBuffered

```
DWORD MIDASgetStreamBytesBuffered(MIDASstreamHandle stream)
```

Gets the number of bytes of stream currently buffered.

Input

stream The stream handle

Description

This function returns the number of bytes of sound data currently stored in the stream buffer. It can be used to monitor the stream playback, and possibly prepare to feed extra data if the figure gets too low.

Return value

The number of bytes of sound data currently buffered.

Operating systems

Win32, Linux

See also

MIDASfeedStreamData

6.3.3 MIDASpauseStream

```
BOOL MIDASpauseStream(MIDASstreamHandle stream)
```

Pauses stream playback.

Input

stream The stream to pause

Description

This function pauses the playback of a stream. When a stream is paused, stream data can be fed normally with *MIDASfeedStreamData*, but nothing will actually be played. Playback can be resumed with *MIDASresumeStream*.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32, Linux

See also

MIDASresumeStream, *MIDASfeedStreamData*

6.3.4 MIDASplayStreamFile

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

Starts playing a digital audio stream from a file.

Input

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. The file must contain raw audio data with no headers — to play WAVE files, use *MIDASplayStreamWaveFile*. The function 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*. A Sound Device channel will be automatically allocated for the stream.

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

MIDASplayStreamWaveFile, MIDASstopStream

6.3.5 MIDASplayStreamWaveFile

```
MIDASstreamHandle MIDASplayStreamWaveFile(char *fileName,
    unsigned bufferLength, int loopStream)
```

Starts playing a digital audio stream from a RIFF WAVE file.

Input

fileName Stream file name

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. The file must be a standard RIFF WAVE (`.wav`) sound file containing raw PCM sound data — compressed WAVE files are not supported. The function 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*. A Sound Device channel will be automatically allocated for the stream.

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

Return value

MIDAS stream handle if successful, NULL if failed.

Operating systems

Win32, Linux

See also

MIDASplayStreamFile, *MIDASstopStream*

6.3.6 MIDASplayStreamPolling

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

Starts playing a digital audio stream in polling mode.

Input

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 *MIDASfeedStreamData*. The stream will continue playing until it is stopped with *MIDASstopStream*. This function will automatically allocate a Sound Device channel for the stream.

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

6.3.7 MIDASresumeStream

```
BOOL MIDASresumeStream(MIDASstreamHandle stream)
```

Resumes stream playback after pause.

Input

stream The stream to resume

Description

This function resumes the playback of a stream that has been paused with *MIDASpauseStream*.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32, Linux

See also

MIDASpauseStream

6.3.8 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

6.3.9 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

6.3.10 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

6.3.11 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, closes the stream file (if playing from a file) and deallocates the stream playback channel.

Return value

TRUE if successful, FALSE if not.

Operating systems

Win32, Linux

See also

MIDASplayStreamFile, *MIDASplayStreamPolling*

Chapter 7

Miscellaneous

7.1 Constants

This section describes all constants used with miscellaneous MIDAS functions. They are grouped according to the `enum` used to define them.

7.1.1 MIDASechoFilterTypes

```
enum MIDASechoFilterTypes
```

Description

These constants define different echo filter types for MIDAS Digital Audio System echo effects. Each echo in an echo effect has a separate filter. If the echo effect contains more than a couple of echoes, the echoes should usually sound best with low-pass filtering.

Values

MIDAS_ECHO_FILTER_NONE No filtering

MIDAS_ECHO_FILTER_LOWPASS Low-pass filtering, higher frequencies are filtered out.

MIDAS_ECHO_FILTER_HIGHPASS High-pass filtering, lower frequencies are filtered out.

7.1.2 MIDASpostProcPosition

```
enum MIDASpostProcPosition
```

Description

These constants define the position of a new post-processor in the system post-processor list when it is being added with *MIDASaddPostProcessor*. The order of the post-processors in the list determines the order in which they are applied to the sound output.

MIDAS internal echo effects are always inserted to the beginning of the list, and filters to the end. Most user post-processors should be inserted to the beginning as well.

Values

MIDAS_POST_PROC_FIRST The post-processor is inserted to the beginning of the list

MIDAS_POST_PROC_LAST The post-processor is inserted to the end of the list

7.2 Data types

This section describes all data types used with miscellaneous MIDAS functions.

7.2.1 MIDASecho

```
typedef struct
{
    unsigned    delay;
    int         gain;
    int         reverseChannels;
    unsigned    filterType;
} MIDASecho;
```

MIDAS echo effect echo structure.

Members

delay Echo delay in milliseconds, in 16.16 fixed point

gain Echo gain, in 16.16 fixed point

reverseChannels 1 if the left and right channels should be reversed in the echo

filterType Echo filter type, see `enum` *MIDASechoFilterTypes*.

Description

This structure defines a single echo of a MIDAS Digital Audio System echo effect. One or more of echoes form an echo set (*MIDASechoSet*), which can then be activated with *MIDASaddEchoEffect*.

The **delay** and **gain** values are given in 16.16 fixed point, which essentially means multiplying the value with 65536. Thus, a delay of 32 milliseconds becomes 2097152 (0x200000), and a gain of 0.5 32768 (0x8000). If **reverseChannels** is 1, data from the left channel is echoed on the right one and vice versa.

7.2.2 MIDASechoHandle

```
typedef ... MIDASechoHandle;
```

Description

MIDASechoHandle is an echo handle that defines an echo effect that is being used.

7.2.3 MIDASechoSet

```
typedef struct
{
    int         feedback;
    int         gain;
    unsigned    numEchoes;
    MIDASecho   *echoes;
} MIDASechoSet;
```

MIDAS echo effect echo set structure.

Members

feedback Echo effect feedback, in 16.16 fixed point

gain Echo effect total gain, in 16.16 fixed point

numEchoes Total number of echoes in the echo effect

echoes Pointer to the echoes

Description

This structure defines a MIDAS Digital Audio System echo set, that can be used with *MIDASaddEchoEffect*. The echo set consists of one or more echoes (*MIDASecho*), plus two controlling variables: **feedback** controls the amount of feedback for the effect, ie. the amount of echoed data that is recycled back to the echo delay line, and **gain** controls the total gain for the effect.

The **feedback** and **gain** values are given in 16.16 fixed point, which essentially means multiplying the value with 65536. Thus, a feedback of 0.8 becomes 52428, and a gain of 0.25 16384 (0x1000). **feedback** typically controls the strength of the echo effect, and is kept at a value below 1, while **gain** can be used to decrease the total volume from the effect to reduce distortion by setting it to a value below 1. Values above 1 are also possible for both **feedback** and **gain**, but should be used with care.

7.2.4 MIDASpostProcessor

```
typedef struct
{
    struct _MIDASpostProcessor *next, *prev;
    void *userData;
    MIDASpostProcFunction floatMono;
    MIDASpostProcFunction floatStereo;
    MIDASpostProcFunction intMono;
    MIDASpostProcFunction intStereo;
} MIDASpostProcessor;
```

MIDAS user post-processor structure.

Members

next, prev, userData Reserved for MIDAS use

floatMono Pointer to the floating-point mono post-processor function, see *MIDASpostProcFunction*

floatStereo Pointer to the floating-point stereo post-processor function, see *MIDASpostProcFunction*

intMono Pointer to the integer mono post-processor function, see *MIDASpostProcFunction*

intStereo Pointer to the integer stereo post-processor function, see *MIDASpostProcFunction*

Description

This structure describes a MIDAS Digital Audio System user post-processor, used with *MIDASaddPostProcessor* and *MIDASremovePostProcessor*. The structure consists of four function pointers, for all possible post-processing functions, plus three reserved members. If any of the post-processor function pointers is NULL, it is simply ignored.

The functions actually used depend on the mixing and output mode, but to be safe all of them should be implemented. The floating-point functions will receive 32-bit floating-point data (C `float`), and the integer ones 32-bit integers (C `int`). The sample data range is normal signed 16-bit range, -32768–32767, but the data has not been clipped yet, so smaller and larger values are also possible — the user should clip them if necessary. The data needs to be modified in place.

Note that it is not necessary to use the post-processing functions to actually modify the data. They could also be used, for example, to gather statistics about the output sample data for

spectrum analyzers. However, remember that the post-processing functions are called in the context of the MIDAS player thread or interrupt. Therefore they may not call MIDAS functions, and should be kept as simple and short as possible.

7.2.5 MIDASpostProcFunction

```
typedef void (MIDAS_CALL *MIDASpostProcFunction)(void *data,
    unsigned numSamples, void *user);
```

Description

A MIDAS user post-processing function, used with in *MIDASpostProcessor* structures. The function receives three arguments: pointer to the sample data to be processed, the number of samples of data to process, and an user pointer passed to *MIDASaddPostProcessor*.

7.3 Functions

This section describes all available miscellaneous MIDAS Digital Audio System functions.

7.3.1 MIDASaddEchoEffect

```
MIDASechoHandle MIDASaddEchoEffect(MIDASechoSet *echoSet);
```

Adds an echo effect to the output.

Input

echoSet The echo set that describes the effect, see *MIDASechoSet*

Description

This function adds an echo effect to the output. An echo effect is described by a *MIDASechoSet* structure, and can consist of one or more echoes. Any number of echo effects can be active simultaneously, the effects added last are processed first. After this function returns, the echo set structure is no longer used by MIDAS and may be deallocated.

Note that echo effects can use large amounts of CPU time, and may even be unusable on 486-class and slower computers.

Return value

MIDAS echo handle for the echo effect if successful, NULL if not.

Operating systems

All

See also

MIDASremoveEchoEffect

7.3.2 MIDASaddPostProcessor

```
BOOL MIDASaddPostProcessor(MIDASpostProcessor *postProc,
    unsigned procPos, void *userData);
```

Adds a user post-processor to the output.

Input

postProc A pointer to a *MIDASpostProcessor* structure that describes the post-processor.

procPos The post-processor position in the post-processor list, see `enum` *MIDASpostProcPosition*.

userData An user data pointer that will be passed to the post-processing functions.

Description

This function adds a user post-processor to the output. The post-processor can be used to alter the output sound data in interesting ways, or to gather information about the data for graphical displays and such. See the *MIDASpostProcessor* structure documentation for more information.

The **postProc** structure may **not** be deallocated or re-used after this function returns, as it is used by MIDAS Digital Audio System internally. The post-processor can be removed from the output with *MIDASremovePostProcessor*, after which the structure can be deallocated.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASremovePostProcessor

7.3.3 MIDASremoveEchoEffect

```
BOOL MIDASremoveEchoEffect(MIDASechoHandle echoHandle);
```

Removes an echo effect from the output.

Input

echoHandle Echo handle for the effect that will be removed

Description

This function removes an echo effect, added with *MIDASaddEchoEffect*, from the output. The echo handle will no longer be valid after this function has been called. Echo effects do not need to be removed in the same order they were added in, but can be added and removed in any order.

Return value

TRUE if successful, FALSE if not

Operating systems

All

See also

MIDASaddEchoEffect

7.3.4 MIDASremovePostProcessor

```
BOOL MIDASremovePostProcessor(MIDASpostProcessor *postProc);
```

Removes a user post-processor from the output.

Input

postProc A pointer to a *MIDASpostProcessor* structure that describes the post-processor.

Description

This function removes a user post-processor added with *MIDASaddPostProcessor* from the output. The post-processing functions will no longer be called, and the post-processor structure may be deallocated.

Return value

TRUE if successful, FALSE if not.

Operating systems

All

See also

MIDASaddPostProcessor