Tapwave® TwMidi Sound API Reference

Version 1.1a

Tapwave TwMidi Sound API Reference

Copyright

© Copyright 2004 Tapwave, Inc. All Rights Reserved. Tapwave is a registered trademark of Tapwave, Inc. The Palm logo, HotSync, Palm OS, Palm, Palm Powered, and the Palm Powered logo are registered trademarks of PalmSource, Inc., and its affiliates. X-Forge is a trademark of Fathammer, Ltd. Java is a registered trademark of Sun Microsystems, Inc. Windows is a registered trademark of Microsoft Corporation, Inc. All other brands are trademarks or registered trademarks of their respective owners.

Background

To use the TwMidi library, your application must include `TwMidi.h`, which is automatically included by `Tapwave.h`.

This library provides facilities for the control and playback of midi. Midi data files (SMF format 0) can be directly played using this API. Midi volume control can also be manipulated. In addition, direct midi synthesizer control (note on, off, etc.) is available. Note that this API is not supported on the simulator.

The device supports a limited number of simultaneous midi voices (16). This means that if a single smf track attempts to use more than 16 simultaneous voices, it will fail.

This document does not attempt to describe the midi standard, the SMF file format. In addition, the actual audible representation of the midi voices are not guaranteed to remain consistent across product revisions. Said another way: If we change the hardware midi chip, then it may sound different in the future.

One major difference between the midi api's here and the PalmOS midi api's is that these api's do not block the application task. For example, when an smf stream is played using TwSmfPlay the call returns immediately and the application can continue execution.

Library Data Types

```
typedef struct TwSmfType* TwSmfHandle;
```

```
typedef struct TwMidiType* TwMidiHandle;
```

API

TwMidiGetLimits

Purpose	Query specific limits to the midi implementation.	

Prototype	`Err TwMidiGetLimits(Int32 aLimitName,`
	`                     Int32* aResult)`	

Parameters	`[in] aLimitName`	The name of the limit to query.
	`[out] aResult`	The value of the limit, assuming aLimitName is a valid limit name.
Result	`errNone` – Succeeded `twMidiErrorNullPointer` – aResult is NULL `twMidiErrorBadParam` – aLimitName is invalid	
Comments	aLimitName must be one of the following values: twMidiLimitMaxSmfHandles – this query returns the maximum number of simultaneous open smf handles supported by TwSmfOpen. twMidiLimitMaxMidiHandles – this query returns the maximum number of simultaneous open midi handles supported by TwMidOpen.	
Side Effects	None.	
Header	`TwMidi.h`	

TwMidiSetMasterVolume

Purpose	Set the master midi volume.	
Prototype	`Err TwMidiSetMasterMidiVolume(Int32 aVolume)`	
Parameters	`[in] aVolume`	The new master midi volume setting. The volume value is in the range of zero to "twMidiMaxVolume" which is defined to be 127.
Result	`errNone` – Succeeded `twMidiErrorBadParam` – aVolume is invalid	
Comments	The maximum volume for midi/smf is 127. Note that this value is different than sndMaxAmp and from the unity gain value (1024) in the sound manager stream api's. The reason is that midi volumes (e.g. key velocities) have the range zero to 127 and it was felt that the midi api's should be self consistent and consistent with the midi standard. To make things easier, two conversion macros are provided: Convert a sound manager volume (0-sndMaxAmp) into a midi volume (0-twMidiMaxVolume): TwMidiCvtSndVolume2MidiVolume(sndvol) Convert a midi volume (0-twMidiMaxVolume) into a sound manager volume (0-sndMaxAmp): TwMidiCvtMidiVolume2SndVolume(midivol)	
Side Effects	The master midi volume is changed for all smf streams and all midi notes being played. Precisely when the volume takes effect is undefined in the sense that how long before it affects any currently playing smf streams or midi notes is undefined. If the volume is changed before a stream is started or a note is turned on then the volume will affect the stream/note. Also note that when your application exits, the master midi volume will be reset to the system default value (which is twMidiMaxVolume).	
Header	`TwMidi.h`	

TwMidiGetMasterVolume

Purpose	Get the master midi volume.	
Prototype	`Err TwMidiGetMasterMidiVolume(Int32* aResult)`	
Parameters	`[out] aResult`	A pointer to where the current master midi volume should be stored.
Result	`errNone` – Succeeded `twMidiErrorNullPointer` – aResult is NULL	
Comments	This call returns the current setting for the midi master volume in the range from 0 to twMidiMaxVolume (127).	
Side Effects	None.	
Header	`TwMidi.h`	

TwMidiPlaySmf

Purpose	Play an SMF encoded block of data.	

Prototype	`Err TwSmfOpen(UInt8* aSMFData,`
	`               UInt32* aDuration,`
	`               Boolean aAsync)`	

Parameters	`[in] aSMFData`	A pointer to SMF Format 0 encoded midi data. This data will be examined by the TwSmfOpen call and if found to be invalid (e.g. not format 0, or other kinds of encoding errors) an error will be returned.
	`[out] aDuration`	A pointer to where the duration, in milliseconds, of the smf data will be stored. This pointer is allowed to be NULL indicating no value will be returned.
	`[in] aAsync`	A flag indicating if the playback should be done synchronously (zero) or asynchronously (non-zero).
Result	`errNone` – Succeeded `twMidiErrorNullPointer` – aResult or aSMFData is NULL `twMidiErrorInvalidFormat` – the smf data is improperly formatted `twMidiErrorAllocFailed` – there are no more smf handles available	
Comments	There are a limited number of smf handles available. Use TwMidiGetLimits to determine the maximum number. This is a handy helper method to play an SMF encoded block of data from beginning to end.	
Side Effects	The aSMFData must remain valid during the playback. It is up to the application to determine how to best do this. Resource data, for example, can be played without issue as long as the application is still running.	
Header	`TwMidi.h`	

TwSmfOpen

Purpose	Create a new TwSmfHandle to an SMF encoded block of data (smf stream).	
Prototype	`Err TwSmfOpen(TwSmfHandle* aResult, UInt8* aSMFData, UInt32* aDuration)`	
Parameters	`[out] aResult`	A pointer to where the newly created TwSmfHandle will be stored.
	`[in] aSMFData`	A pointer to SMF Format 0 encoded midi data. This data will be examined by the TwSmfOpen call and if found to be invalid (e.g. not format 0, or other kinds of encoding errors) an error will be returned.
	`[out] aDuration`	A pointer to where the duration, in milliseconds, of the smf data will be stored. This pointer is allowed to be NULL indicating no value will be returned.
Result	`errNone` – Succeeded `twMidiErrorNullPointer` – aResult or aSMFData is NULL `twMidiErrorInvalidFormat` – the smf data is improperly formatted `twMidiErrorAllocFailed` – there are no more smf handles available	
Comments	There are a limited number of smf handles available. Use TwMidiGetLimits to determine the maximum number.	
Side Effects	The smf handle is allocated and reserved for future playing. Playback does not begin until TwSmfPlay is called.	
Header	`TwMidi.h`	

TwSmfClose

Purpose	Close and free an existing smf handle.	
Prototype	`Err TwSmfClose(TwSmfHandle aHandle)`	
Parameters	`[in] aHandle`	The handle to the smf stream to close.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is not a valid smf handle	
Comments	If the smf stream was playing, the playback is stopped before freeing the smf handle.	
Side Effects	None.	
Header	`TwMidi.h`	

TwSmfPlay

Purpose	Start playback on an smf stream.	

Prototype	`Err TwSmfPlay(TwSmfHandle aHandle,`
	`              SndSmfOptionsType* aOptions,`
	`              SndSmfChanRangeType* aRange,`
	`              SndCallbackInfoType* aCallback)`	

Parameters	`[in] aHandle`	The handle to the smf stream to play.
	`[in] aOptions`	Optional pointer to playback options. See SoundMgr.h for the definition. If NULL is passed in then the entire stream will be played at maximum amplitude.
	`[in] aRange`	Optional pointer to channel range definition. See SoundMgr.h for the definition. If NULL is passed in then all channels will be played.
	`[in] aCallback`	Optional pointer to a callback object. See SoundMgr.h for the definition. The type of the callback function "aCallback.funcP" is: `void (*)(UInt32);` The argument to the callback function is "aCallback.dwUserData".
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is not a valid smf handle `twMidiErrorAlreadyPlaying` – the smf stream is already playing	
Comments	The aOptions, aRange and aCallback arguments are optional. The only safe thing to do in the callback function is to call TwSmfPlay to achieve seemless looping. **Any** other OS, Library, or application call will have undefined and likely disastrous results. Also note that the playback is begun with this call. The application task is not blocked unlike the PalmOS midi api calls.	

Side Effects	When the playback is completed the callback will be invoked. Note that this is the only condition where the callback is invoked.
Header	`TwMidi.h`

TwSmfIsPlaying

Purpose	Query the playback status of the smf stream.	

Prototype	`Err TwSmfIsPlaying(TwSmfHandle aHandle,`
	`                   Boolean* aIsPlaying)`	

Parameters	`[in] aHandle`	The handle to the smf stream to query.
	`[out] aIsPlaying`	Pointer to a Boolean that will be set to the current playback status of the stream.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is not a valid smf handle `twMidiErrorNullPointer` – aIsPlaying is NULL	
Comments	If the smf stream is playing at the time of the call then *aIsPlaying is set to true, otherwise false. Note that the stream may stop playing at any time, including (conceptually) in the middle of this call. However, the atomicity of the call is guaranteed.	
Side Effects	None.	
Header	`TwMidi.h`	

TwSmfStop

Purpose	Stop playback on an smf stream.	
Prototype	`Err TwSmfStop(TwSmfHandle aHandle)`	
Parameters	`[in] aHandle`	The handle to the smf stream to stop playing.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is not a valid smf handle	
Comments	If the smf stream is playing then it is stopped; there is no error returned if the stream was not already playing.	
Side Effects	None.	
Header	`TwMidi.h`	

TwMidiOpen

Purpose	Create a new midi handle.	
Prototype	`Err TwMidiOpen(TwMidiHandle* aResult)`	
Parameters	`[out] aResult`	A pointer to where the newly created TwMidiHandle will be stored.
Result	`errNone` – Succeeded `twMidiErrorNullPointer` – aResult is NULL `twMidiErrorAllocFailed` – there are no more midi handles available	
Comments	A midi handle is used to do direct access to the midi synthesizer (e.g. turn on/off individual notes, change voices, etc.). There are a limited number of midi handles available. Use TwMidiGetLimits to determine how many are available.	
Side Effects	None.	
Header	`TwMidi.h`	

TwMidiClose

Purpose	Close and destroy a previously created midi handle.	
Prototype	`Err TwMidiClose(TwMidiHandle aHandle)`	
Parameters	`[in] aHandle`	The handle to the previously opened midi synthesizer.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is invalid	
Comments	The handle is closed and if the handle was playing anything, the playing is stopped.	
Side Effects	None.	
Header	`TwMidi.h`	

TwMidiNoteOn, TwMidiNoteOff

Purpose	Turn a midi channel on or off.	

Prototype	`Err TwMidiNote[On,Off](TwMidiHandle aHandle,`
	`UInt8 aChannel,`
	`UInt8 aKey,`
	`UInt8 aVelocity)`	

Parameters	`[in] aHandle`	The handle to the previously opened midi synthesizer.
	`[in] aChannel`	The midi channel number to turn on/off. The channel number must be in the range of zero to 15, inclusive.
	`[in] aKey`	The midi key (note) to stop/start playing.
	`[in] aVelocity`	The velocity (volume) of the key to stop/start playing.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is invalid `twMidiErrorBadParam` – aChannel, aKey or aVelocity are invalid	
Comments	Note that these calls do not take a duration. It is up to the caller to simulate a duration by timing the calls to these functions.	
Side Effects	None.	
Header	`TwMidi.h`	

TwMidiProgramChange

Purpose	Change the "program" or voice assigned to a midi handle.	
Prototype	`Err TwMidiProgramChange(TwMidiHandle aHandle, UInt8 aChannel, UInt8 aProgram)`	
Parameters	`[in] aHandle`	The handle to the previously opened midi synthesizer.
	`[in] aChannel`	The midi channel number to change.
	`[in] aProgram`	The new program (voice) to use.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is invalid `twMidiErrorBadParam` – aChannel or aProgram are invalid	
Comments	Change the program (voice) used by the midi handle. Note that the program change will be silently ignored for channel 9 (the drum channel).	
Side Effects	None.	
Header	`TwMidi.h`	

TwMidiControlChange

Purpose	Perform a control change on the midi handle.	

Prototype	`Err TwMidiControlChange(TwMidiHandle aHandle,`
	`UInt8 aChannel,`
	`Int32 aControl,`
	`Int32 aValue)`	

Parameters	`[in] aHandle`	The handle to the previously opened midi synthesizer.
	`[in] aChannel`	The midi channel number to change.
	`[in] aControl`	The control to change.
	`[in] aValue`	The value of the control.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is invalid `twMidiErrorBadParam` – aChannel, aControl or aValue are invalid	

Comments	This will effect a control change on the given channel. The support control change values are:
	
	`0x01 – modulation`
	`0x07 – channel volume`
	`0x0A – panpot`
	`0x78 – all sound off (aValue is ignored)`
	`0x79 – reset all controllers (aValue is ignored)`
	`0x7B – all notes off (aValue is ignored)`
	`0x65 – rpn msb`
	`0x64 – rpn lsb`
	`0x63 – Nrpn msb`
	`0x62 – Nrpn lsb`
	`0x06 – data entry msb`
	
	Please see the midi specification for more information on what these mean.	

Side Effects	None.
Header	`TwMidi.h`

TwMidiPitchBend

Purpose	Set the pitch bend for a given midi channel.	
Prototype	`Err TwMidiPitchBend(TwMidiHandle aHandle, UInt8 aChannel, Int32 aValue)`	
Parameters	`[in] aHandle`	The handle to the previously opened midi synthesizer.
	`[in] aChannel`	The midi channel number to change.
	`[in] aValue`	The value of the bend.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is invalid `twMidiErrorBadParam` – aChannel or aValue are invalid	
Comments	This will effect a pitch bend on the given channel.	
Side Effects	None.	
Header	`TwMidi.h`	

TwMidiSysEx

Purpose	Execute a system exclusive message on the midi channel.	

Prototype	`Err TwMidiSysEx(TwMidiHandle aHandle,`
	`UInt8 aChannel,`
	`UInt8* aData,`
	`UInt16 aSize)`	

Parameters	`[in] aHandle`	The handle to the previously opened midi synthesizer.
	`[in] aChannel`	The midi channel number to change.
	`[in] aData`	The system exclusive message data, in midi file format.
	`[in] aSize`	The number of bytes of message data.
Result	`errNone` – Succeeded `twMidiErrorInvalidHandle` – aHandle is invalid `twMidiErrorBadParam` – aChannel or aValue are invalid	
Comments	This will effect a system exclusive message on the given channel.	
Side Effects	None.	
Header	`TwMidi.h`	

Examples

This simple example plays a midi file located on a card (error handling is an exercise left to the reader):

```
void playMidiFile(Int32 Volume, Int32 VolRefNum, char* Path) {
    TwMidiSetMasterVolume(Volume);

    unsigned char* midiData = readMidiData(VolRefNum, Path);
    if (midiData) {
        UInt32 duration;
        TwSmfType* smfHandle;
        Err err = TwSmfOpen(&smfHandle, midiData, &duration);
        if (!err) {
            err = TwSmfPlay(smfHandle, NULL, NULL, NULL);
            if (!err) {
                // Wait for playback to finish. Note that this is an ARM
                // example, therefore the units to SysTaskDelay are in
                // milliseconds not centiseconds.
                SysTaskDelay(duration + 200);
            }
            TwSmfClose(smfHandle);
        }
        delete midiData;
    }
}

unsigned char* readMidiData(Int32 VolRefNum, char* Path) {
    UInt32 size, nb;
    FileRef ref;
    unsigned char* buf = NULL;
    Err err = VFSFileOpen(VolRefNum, Path, vfsModeRead, &ref);
    if (!err) {
        err = VFSFileSize(ref, &size);
        if (!err) {
            buf = new unsigned char[size];
            if (buf) {
                err = VFSFileRead(ref, size, buf, &nb);
            }
        }
        VFSFileClose(ref);
    }
    return buf;
}
```