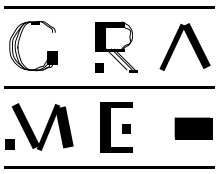

MidiShare Functions Library

MIDI Files Management

GRAME Research Lab.
6 quai Jean Moulin - BP 1185
69202 LYON CEDEX 01
Ph: (33) 72.07.37.00 Fax: (33) 72.07.37.01

e-mail : GRAME@rd.grame.fr

GRAME

Functions Summary

MidiFileGetVersion	return the version numbers
MidiFileOpen	open an existing MIDI file
MidiFileClose	close a file opened with MidiFileOpen
MidiFileCreate	create a MIDI file
MidiFileOpenTrack	open an existing track
MidiFileNewTrack	create a new track
MidiFileCloseTrack	close a track
MidiFileSetPos	locate to the beginning of a track
MidiFileReadEv	read an event within the current track
MidiFileReadTrack	read a track
MidiFileWriteEv	write an event to the current track
MidiFileWriteTrack	write a sequence to a track
MidiFileWriteEv	write an event to the current track
MidiFileWriteTrack	write a sequence to a track
MidiFileGetMFErrno	returns the MidiFile_errno code
MidiFileGetErrno	returns the errno code

Warning: all these functions doesn't check for datas consistency, according to the MIDIFile specifications.

MidiFileChooseTrack

Function

Synopsis

```
#include <MidiFile.h>
Boolean MidiFileChooseTrack( midiFILE *fd, short numTrack);
```

Description

Locate at the beginning of the track *numTrack* within the file pointed to by *fd*.
An error occur if the function returns false:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

MidiFileClose

Function

Synopsis

```
#include <MidiFile.h>
Boolean MidiFileClose( midiFILE *fd);
```

Description

Close the file pointed to by *fd*, previously opened with MidiFileOpen or MidiFileCreate. If a track is still opened, the function closes it with a call to MidiFileCloseTrack.

see also : MidiFileOpen MidiFileCreate

MidiFileCloseTrack

Function

Synopsis

```
#include <MidiFile.h>
Boolean   MidiFileCloseTrack( midiFILE *fd);
```

Description

Close a track previously opened with MidiFileOpenTrack or created with MidiFileNewTrack.

- If the file is opened for reading, MidiFileCloseTrack locate the file pointeur at the beginning of the next track.
- If the file is opened for writing, MidiFileCloseTrack flush the KeyOff sequence (coming from typeNote events), update the track header and the file header.

The function does nothing and returns true if the track is still closed.

An error occur if the function returns false:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

see also : MidiFileOpenTrack MidiFileNewTrack

MidiFileCreate

Function

Synopsis

```
#include <MidiFile.h>
midiFILE *MidiFileCreate( const char *filename, short format, short timeDef, short ticks);
```

Description

Create a MIDIFile format file. The function parameters are as follow:

filename : name of the file to create.

format : MIDIFile format of the file, it can takes the following values:
- midifile0 : format 0 (one track)
- midifile1 : format 1 (several tracks, to read according to the tempo map contained in the track #0)
- midifile2 : format 2 (several independant patterns one per track, every track contains its own tempo map)

timeDef : specify the time representation, it can takes the following values:
- TicksPerQuarterNote : MIDI measured time.
- Smpte24 : smpte time 24 frame/sec.
- Smpte25 : smpte time 25 frame/sec.
- Smpte29 : smpte time 29 frame/sec.
- Smpte30 : smpte time 30 frame/sec.

ticks : for MIDI time: represents the ticks count per quarter note.
for smpte time: represents the ticks count per frame.

The function returns a pointer to the following structure:

```
typedef struct midiFILE{
        short         format;                   /* file format                          */
        unsigned short ntrks;   /* track count                          */
        short         time;                     /* time representation :                */
                                                /* for MIDI time: tick count per quarter note      */
                                                /* for smpte time: b. 15  = 1                  */
                                                /*              b.8-14 = frame count per sec   */
                                                /*                       b.0-7  = tick count per frame  */
        FILE          *fd;                      /* standard file descriptor             */
        fpos_t  trkHeadOffset;  /* track header offset                  */
                                                /* nil if the track is closed           */
        long          _cnt;
        MidiSeqPtr keyOff;      /* keyOff coming from typeNote events   */
        long          curDate;    /* current date                         */
        Boolean mode;                           /* 0/1 : reading/writing                */
}midiFILE;
```

An error occur if the function returns null:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

see also : MidiFileOpen MidiFileClose

MidiFileGetVersion

Function

Synopsis

```
#include <MidiFile.h>
const MDF_versions *MidiFileGetVersion(void);
```

Description

Return the version number of the implemented MIDIFile format and of the source code.

MidiFileGetVersion returns a pointer on the following structure:

```
typedef struct MDF_versions{
        short         src;                                  /* source code version          */
        short         MidiFile;                     /* MIDIFile format version      */
```

```
}MDF_versions;
```

MidiFileNewTrack

Function

Synopsis

```
#include <MidiFile.h>
Boolean  MidiFileNewTrack( midiFILE *fd);
```

Description

MidiFileNewTrack adds a new track header at the end of the file and open the corresponding track. You can use this function only if the file is opened for writing. A previously opened track will first been closed.

An error occur if the function returns false:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

see also : MidiFileOpenTrack MidiFileCloseTrack

MidiFileOpen

Function

Synopsis

```
#include <MidiFile.h>
midiFILE *MidiFileOpen( const char *filename, short mode);
```

Description

Open an existing MIDI file. The function parameters are as follow:

filename : name of the file.
mode : can takes the following values :
- MidiFileRead : to read the file
- MidiFileAppend : to append to the file

The function returns a pointer to the following structure:

```
typedef struct midiFILE{
        short         format;                 /* file format                          */
        unsigned short ntrks;   /* track count                          */
        short         time;                   /* time representation :                */
                                              /* for MIDI time: tick count per quarter note      */
                                              /* for smpte time: b. 15  = 1                  */
                                              /*              b.8-14 = frame count per sec   */
                                              /*                       b.0-7  = tick count per frame  */
        FILE          *fd;                    /* standard file descriptor             */
        fpos_t  trkHeadOffset;  /* track header offset                  */
                                              /* nil if the track is closed           */
        long          _cnt;
        MidiSeqPtr keyOff;      /* keyOff coming from typeNote events   */
        long          curDate;  /* current date                         */
        Boolean mode;                         /* 0/1 : reading/writing                */
}midiFILE;
```

An error occur if the function returns nil:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

see also : MidiFileClose MidiFileCreate

MidiFileOpenTrack

Function

Synopsis

```
#include <MidiFile.h>
Boolean  MidiFileOpenTrack( midiFILE *fd);
```

Description

Open the track if the file is opened for reading, otherwise the function returns false and MidiFile_errno is set to MidiFileErrNoTrack. The function does nothing and returns true if the track is still opened. The purpose of this function consists essentially in data initialization to facilitate the track handling.

see also : MidiFileNewTrack MidiFileCloseTrack

MidiFileReadEv

Function

Synopsis

```
#include <MidiFile.h>
MidiEvPtr  MidiFileReadEv( midiFILE *fd);
```

Description

MidiFileReadEv returns the next event within the current track. The track must be opened using MidiFileOpenTrack before reading an event. When you reach the end of the current track, it is automaticaly closed and the function returns nil.

An error occur if the function returns nil when the track is still opened:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

see also : MidiFileReadTrack isTrackOpen

MidiFileReadTrack

Function

Synopsis

```
#include <MidiFile.h>
MidiSeqPtr MidiFileReadTrack( midiFILE *fd);
```

Description

The function reads the current track from the file and returns the result in a MidiShare sequence. MidiFileReadTrack automaticaly opens and closes the track to read.

An error occur if the function returns nil when the track is still opened:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

MidiFileWriteEv

Function

Synopsis

```
#include <MidiFile.h>
Boolean    MidiFileWriteEv( midiFILE *fd, MidiEvPtr ev);
```

Description

MidiFileWriteEv writes the event *ev* to the current track. The track must be previously opened using the MidiFileNewTrack function.

An error occur if the function returns false:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

see also : MidiFileWriteTrack

MidiFileWriteTrack

Function

Synopsis

```
#include <MidiFile.h>
Boolean MidiFileWriteTrack( midiFILE *fd, MidiSeqPtr seq);
```

Description

Writes in order all the events of the sequence *seq* to the file pointed to by *fd*. MidiFileWriteTrack automaticaly create and close the written track.

An error occur if the function returns false:
it is a MidiShare error if MidiFile_errno is not equal to MidiFileNoErr,
otherwise, the error code is in errno.

see also : MidiFileWriteEv

MidiFileGetMFErrno

Function

Synopsis

```
#include <MidiFile.h>
int MidiFileGetMFErrno( void);
```

Description

returns the MidiFile_errno code

MidiFileGetErrno

Function

Synopsis

```
#include <MidiFile.h>
int MidiFileGetErrno( void);
```

Description

returns the MidiFile_errno code

isTrackOpen

macro

```
#include <MidiFile.h>

#ifdef __cplusplus
        inline Boolean isTrackOpen( midiFILE *fd)
                                                { return (fd->trkHeadOffset> 0) }
#else
        #define isTrackOpen(fd)         (fd->trkHeadOffset > 0)
#endif
```

Description

Returns the current track state: closed or opened.

Source code examples

A typical example of code to read a MIDI file might be the following one:

```
MidiSeqPtr ReadMIDIFile( char *itsName)
{
	MidiSeqPtr seq, tmp;
	midiFILE *fd;
	unsigned short n;

	seq= MidiNewSeq()                                   /* allocate a new MidiShare sequence */
	if( fd= MidiFileOpen( itsName, MidiFileRead))
	{
		n= fd->ntrks;                                       /* get the number of tracks          */
		while( n--) {
			tmp= MidiFileReadTrack( fd);        /* read every track                  */

			Mix( tmp, seq);                                 /* the Mix function is to provide*/
                                                            /* it transfers the content of the first  */
                                                            /* sequence to the second one,            */
                                                            /* its interface might be :               */
                                                            /* void Mix( MidiSeqPtr src, MidiSeqPtr dst)*/

			MidiFreeSeq( tmp);      /* this sequence is now empty, we can        */
                                                            /* free it without freing the readed  */
                                                            /* events                             */
		}
		MidiFileClose( fd);
	}
	return seq;
}
```

A typical example of code to create a format 1 MIDI file might be the following one:

```
void WriteMIDIFile( char *itsName)
{
	midiFILE *fd;

	/* we first create a new MIDI file using a format 1          */
	if( fd= MidiFileCreate( itsName, midifile1, TicksPerQuarterNote, 500))
	{
		/* for the file consistency, the first track         */
		/* to write is the tempo map                                 */
		MidiFileWriteTrack( fd, myTempoMap);

		/* then we can write all the other tracks            */
                                                /*
                                                  it is the program responsibility to determine
                                                  the content of the tracks. Here, every track is
                                                  stored in separate MidiShare sequences (myTempoMap,
                                                  track1, track2,...trackn). They are supposed to be
                                                  global variables. Of course, events in the file will
                                                  keep exactly the same order than in the sequence
                                                */
		MidiFileWriteTrack( fd, track1);
		MidiFileWriteTrack( fd, track2);
		/* ...*/
		MidiFileWriteTrack( fd, trackn);

		/* and we finaly close the file                              */
		MidiFileClose( fd);
	}
}
```

Warning! take care of that these examples doesn't check for errors