INSIDE MAD LIBRARY 5.2

by Antoine ROSSET 98-99
16 BD Tranchées
1206 GENEVA
Switzerland

Available for CodeWarrior – C/C++ Compilers - MPW

This is a cross-platform documentation : for MacOS, BeOS and Windows95/NT.

MADLibrary is a collection of routines that you can use to implement music in your applications. You can use MADLibrary to...

- play music
- play sounds during music
- read different formats available in plugs : MAD, MOD, XM, S3M, MTM, etc.
- create your own soundtracker editor or player

Modifications since version 5.0 (PlayerPRO 5.0)

- MADDisposeMusic now requires the current driver in which the music is used. If it is not attached to a driver, you can pass NIL.
- There is a new option in driver: oversampling.

Modifications since version 4.6 (PlayerPRO 4.5.9)

- On PPC you will need SoundLib (included in your compiler folder). You have to include it as a Import Weak ! or it will crash!
- This new release simplified the driver, less options but better sound quality.
- MADLibrary supports now ONLY DeluxeStereoOutPut mode!
- General volume control functions added: Mac Hardware volume and Driver Software volume.
- Plugs can now fully support application or system memory allocation.
- If you don't want to to include Plugs files with your project, you can include them as resources (ONLY in the application resource):

  - start ID : 5000

  - At least : CODE and STR# (with same ID), optional : PPCC

See "example project" and "InternalPlug.Rsrc" file.

You can add more plugs: 5001, 5002, etc...

Modifications since last version 4.5 (PlayerPRO 4.5.3)

- A new settings option : TickRemover
- MADPlay() has been renamed MADStartDriver()
- MADStop() has been renamed MADStopDriver()
- MADDriver->reading = true has been replaced by MADPlayMusic()
- MADDriver->reading = false has been replaced by MADStopMusic()
- MADDriver structure is now in STATIC memory, to access it use MADGetMADDriverPtr()

- MADLoadMusicFile(name) has been replace by MADLoadMusicFile("MADH", name)

- MADImportMusicFile has been deleted (use MADLoadMusicFile).

- Some OSType have been replaced by char*

- Some Str255 pascal string have been replaced by C String char*

- If you are using INTERNAL Plugs (resources), you need to update them with new plugs available in PlayerPRO.

Modifications since last version 4.5 (PlayerPRO 4.5.5)

- MADGetMusicStatus in Ticks (1/60th sec)! NOT in secs anymore !

- A new function MADSetMusicStatus()

Modifications in version 4.6 (PlayerPRO 4.6)

- You can now have multiple musics in memory. You simply Attach them to a driver with MADAttachDriverToMusic

- You can now have multiple drivers in memory... it means you can play two musics at the same time in the same application.

MADLibrary Installation

To use the MADLibrary functions, install the library in your project and make an #include”RDriver.h” in your file “.c”. You'll need 3 files: MAD.h, RDriver.h and MADLibrary.

MADLibrary Reference

This section serves as a reference to the routines MADLibrary provides.

MADInitLibrary

This function initializes the MADLibrary package.

```
OSErr MADInitLibrary(char *PlugsFolderName, Boolean UseSystemMemory,
                     MADLibrary **MADLib);
```

DESCRIPTION

The MADInitLibrary function is used to initialize the MADLibrary package. MADInitLibrary performs some checks to see is MADLibrary can run, and then sets up some internal data structures in MADLib (it allocates about 20kb). You must call MADInitLibrary before calling any other MADLibrary routine. It will also check if there are some Import/Export plugs available: it checks application directory and the PlugsFolderName directory if it exists: you can give a "" filename, but not a 0L ! In general you should use "Plugs" for Plugs Folder.

UseSystemMemory is a boolean value that indicates if you want to load plugs in system heap or application heap, for a normal usage UseSystemMemory should be set to false.

MADInitLibrary returns an error code is initialization fails, other wise it returns noErr.

Keep MADLib in a global variable, so you will be able to use it everywhere in your application.

MADDisposeLibrary

This function shuts down the MADLibrary package.

```
void MADDisposeLibrary(MADLibrary *MADLib);
```

DESCRIPTION

The `MADDisposeLibrary` function is used to shut down the MADLibrary package. It should be called when the application is finished using MADLibrary to balance the original call to `MADInitLibrary`. At this point no further calls should be made to any MADLibrary routines until `MADInitLibrary` is called again.

MADGetBestDriver

This will check current Mac hardware and fill the MADDriverSettings structure with the best settings for current Mac. This function doesn't call any other functions of MADLibrary.

```
OSErr MADGetBestDriver( MADDriverSettings *driverParam);
```

driverParam	A pointer to your driver settings:
numChn	Active tracks, automatically updated when a new music is loaded.
outPutBits	8 or 16 bits
outPutRate	Fixed number, by example: rate44Khz, rate22050khz, rate11khz, etc...
outPutMode	MUST be DeluxeStereoOutPut
driverMode	SoundManager, BeOS or Win95/NT
repeatMusic	When music will be over, repeat it?
sysMemory	Allocate memory in application heap(false) or in system heap(true).
MicroDelaySize	Micro delay duration (in ms, max 1 sec = 1000 ms)
surround	Surround effect active?
Reverb	Reverb effect active?
ReverbSize	Delay between echos in ms
ReverbStrength	Strength of reverb in %
Oversampling	A value from 1 to 20, 1 means NO oversampling

DESCRIPTION

This will check current Mac hardware and fill the MADDriverSettings structure with the best settings for current Mac. This function doesn't call any other functions of MADLibrary. The common usage is to use it just before MADCreateDriver function.

MADCreateDriver

This function will create a new music driver, allowing you to specify the settings to be used.

```
OSErr MADCreateDriver(    MADDriverSettings *driverParam, MADLibrary
                          *MADLib, MADDriverRec** returnDriver);
```

See MADGetBestDriver function for informations about `MADDriverSettings *driverParam.`

This function returns the created driver in `returnDriver.`

DESCRIPTION

You have to call this function before calling loading and playing functions. The `MADCreateDriver` function is used to create a new music driver. It is strongly advised to launch this routine at the beginning of your program. See example.c to see parameters how you can perform an automatic set up of driverParam by using `MADGetBestDriver`. This function allocates about 10kb. You have to call `MADDisposeDriver` if you want to free memory.

MADDisposeDriver

This function deletes current music driver created with `MADCreateDriver`.

```
OSErr MADDisposeDriver(MADDriverRec *MDriver);
```

DESCRIPTION

This function deletes current music driver created with `MADCreateDriver`. You cannot use loading and playing function after this call (you have to call `MADCreateDriver` again.)

MADMusicIdentifyCString /MADMusicIdentifyPString / MADMusicIdentifyFSp

This will identify what kind / format of music is a file.

```
OSErr MADMusicIdentifyPString (MADLibrary *, char *type, Str255 name);

Or

OSErr MADMusicIdentifyCString (MADLibrary *, char *type, Ptr name);

Or

OSErr MADMusicIdentifyFSp (MADLibrary *, char *type, FSSpec *theSpec);
```

`name`	Music file name in current directory.
type	File format of this music. If it returns an error, type = "!!!!"

DESCRIPTION

The `MADMusicIdentify` function is used to identify a music file.

MADLoadMusicRsrc

This will load a music resource into memory. The music resource has to be a 'MADI' music format, created with PlayerPRO last version.

```
OSErr MADLoadMusicRsrc( MADMusic **music,
                        OSType resType,
                        short resID);
```

`music`	The music itself
`resType`	Resource type
resID	Resource ID

DESCRIPTION

The `MADLoadMusicRsrc` function is used to load a PlayerPRO music resource in memory. You have to open your resource file before if resource is not in application resources, otherwise it will return a file error.

How to convert resources <-> files

There is a hidden feature in PlayerPRO to do that. You'll need PlayerPRO and ResEdit.

Resource to file:

Open your resource file with ResEdit and COPY the resource in the clipboard.

Open PlayerPRO and select the MusicList Window.

Press on your space bar AND PASTE.

PlayerPRO will ask you where to save the file. The type of the file will be resource type.

File to resource:

Add your music file to PlayerPRO Music List Window and select it.

COPY it in clipboard.

Clipboard now contains your music file with the same type as your music file type.

PASTE it in ResEdit.

(MADLoadMusicRsrc supports ONLY MADI music format!)

If you want to change the resource type, change the music file by pressing '?' button in PlayerPRO and then COPY it.

You can also easily create 'MADI' resource by exporting your music as an application.

MADLoadMusicPtr

This will load a music pointer into memory. The music pointer has to be a 'MADH' file, created with PlayerPRO last version.

```
OSErr MADLoadMusicPtr( MADMusic **music, Ptr musicPtr);
```

`music`	The music itself
`musicPtr`	A pointer on music data

DESCRIPTION

The `MADLoadMusicPtr` function is used to load a PlayerPRO music pointer in memory. You can dispose your pointer after this call, MADLibrary will not access data on your pointer.

MADLoadMusicFileCString / MADLoadMusicFilePString / MADLoadMusicFSpFile

This will load a music file into memory. This function will check if there is a 'Plug-ins' to load this music.

```
OSErr MADLoadMusicFilePString(   MADMusic **music,
                                 char *type,
                                 Str255 name);
```

or

```
OSErr MADLoadMusicFileCString(   MADMusic **music,
                                 char *type,
                                 Ptr name);
```

or

```
OSErr MADLoadMusicFSpFile(       MADMusic **music,
                                 char *type,
                                 FSSpec *theSpec);
```

`OSType`	File type : "MADH", "MADF", "XM " ,"S3M ", etc.

DESCRIPTION

The `MADLoadMusic` function is used to load a PlayerPRO music file into memory. It will check if there are some Import/Export plugs available for this type of music: it checks application directory and the 'Plugs' directory if it exists.

MADAttachDriverToMusic

This function will attach a music pointer to a driver.

```
OSErr MADAttachDriverToMusic (   MADDriverRec *driver,
                                 MADMusic *music);
```

`MADDriverRec`	the music driver
`MADMusic`	the music

DESCRIPTION

The `MADAttachDriverToMusic` function is used to attach a music to a driver. You need to do this to play this music. You don’t need to detach this music, it will be automaticaly detached when the driver is destroyed. You can attach the same music to 2 or more drivers.

MADDisposeMusic

This will dipose current music in memory after a load function.

```
OSErr MADDisposeMusic( MADMusic **music, MADDriverRec *driver);
```

DESCRIPTION

The `MADDisposeMusic` function is used to dispose a music pointer. You have to pass the current driver in which the music is attached. If the music is not attached to a driver, you can pass NIL.

MADStartDriver

This will activate the current sound generating procedure defined by the driver.

```
OSErr MADStartDriver(MADDriverRec *MDriver);
```

DESCRIPTION

The `MADStartInterruption`function is used to activate the.

MADStopDriver

This will desactivate current sound generating procedure.

```
OSErr MADStopDriver(MADDriverRec *MDriver);
```

DESCRIPTION

The `MADStopInterruption`function is used to stop playing the current music in memory.

MADSetMusicStatus

This will change current position of current music in memory.

```
OSErr MADSetMusicStatus( MADDriverRec *Mdriver, long min, long max,
long cur);
```

DESCRIPTION

By example to set the music to the middle : `MADSetMusicStatus( 0, 100, 50);`

MADGetMusicStatus

This will get informations about position and full duration of current music in memory.

```
OSErr MADGetMusicStatus( MADDriverRec *Mdriver, long *fullTime, long
*curTime);
```

DESCRIPTION

Values are in seconds. You can use toolbox function Secs2Date to convert them in hours, minutes and seconds.

MADReset

This will reset reading position of current music in memory to startup position.

```
OSErr MADReset( MADDriverRec *MDriver);
```

DESCRIPTION

The `MADReset` function is used to reset reading position of current music in memory to startup position.

MADPlaySndHandle

This will play a sound handle of 'snd ' type on a MADLibrary driver channel.

```
OSErr MADPlaySndHandle (   MADDriverRec *Mdriver,
                           Handle sndRsrc,
                           long channel,
                           long note);
```

`sndRsrc`	Handle on a 'snd ' handle (see Inside Macintosh)
`channel`	Channel ID which will be used to play this sound
`note`	note ID: from 0 (C0) to 95 (B7) or 0xFF (play this sound at normal rate)

DESCRIPTION

The `MADPlaySndHandle` function is used to play a sound 'snd ' on a MADLibrary driver channel. WARNING: This function will change the sndRsrc handle, you will not be able to use it with normal SoundManager functions: you will have to reload it. (It will inverse amplitude of data for faster processing).

MADPlaySoundData

This will play a sound data on a MADLibrary driver channel. If you want to play a snd resource, use MADPlaySndHandle function instead of this one.

```
OSErr MADPlaySound( MADDriverRec *MDriver, Ptr soundData, long
                   soundDataSize, long channel, long note, long amp,
                   long loopBegin, long loopSize, unsigned long
                   rate);
```

`soundData`	Pointer on raw data
`soundDataSize`	Sound size
`channel`	Channel ID which will be used to play this sound
`note`	note ID, 0xFF : play this sound at normal rate
`amp`	amplitude of this sound, 8 or 16
`loopBegin`	Loop begin
`loopSize`	Loop size
`rate`	sample rate of this sound data

DESCRIPTION

The `MADPlaySound` function is used to play a sound data pointer on a MADLibrary driver channel. The function MADPlaySoundDataSYNC will allow to play a sound in syncronous mode, it will return only when the sound is finished.

MADGetHardwareVolume

This function returns the Mac HARDWARE volume, not SOFTWARE volume. To get SOFTWARE volume, use MADDriver->VolGlobal (see RDriver.h). Minimum volume = 0 (0%), Maximum volume = 64 (100%).

```
long MADGetHardwareVolume(void);
```

DESCRIPTION

The `MADGetHardwareVolume` function uses `GetSoundVol` or `GetDefaultOutputVolume` functions from ToolBox.

MADSetHardwareVolume

This function changes the Mac HARDWARE volume, not SOFTWARE volume. To change SOFTWARE volume, use MADDriver->VolGlobal (see RDriver.h). Minimum volume = 0 (0%), Maximum volume = 64 (100%).

```
OSErr MADSetHardwareVolume(long volume);
```

`volume`	Volume value: from 0 to 64

DESCRIPTION

The `MADSetHardwareVolume` function uses `SetSoundVol` or `SetDefaultOutputVolume` functions from ToolBox.