

PLAYING XM MUSIC FROM EXE

Date: 04 September 2004
Author: Fornix (Group **FUSION**)
Email: fornixrulez@yahoo.com
Website: <http://fusionrulez.cjb.net>
Target: No targets today
Audience: novice ASM Programmer
Tools: MASM32 Compiler, MSDN LIBRARY (Even a programmer's WIN32API reference or online MSDN reference www.msdn.microsoft.com will do)
A simple search for masm32 at Google will fetch you results to download MASM32 package

DISCLAIMER

Information contained in this tutorial is for educational purposes only. I am not responsible for what you may do after reading this tutorial.

INTRODUCTION

I was always very much attracted to the music which I had heard in some KeyGens. I noticed that there were hardly any essays explaining the method of writing such a program to play XM music in ASM. In one of the Code Breakers Magazine I found a tutorial which explained the procedure of playing music in C++. It was very complicated. But life in ASM is much simpler than C++ (Exclaimed?). In this tutorial, we'll make a simple Message Box which is very much similar to Iczelion's First ASM Tutorial. The only difference is the addition of music to add spice to it. I would like to thank my friend Blue_Orka for providing source codes of programs which played XM music. It is through this that I learned this technique.

TUTORIAL

I assume that you know something about assembly programming. A pure newbie will find this haunting! Don't worry if you don't know Assembly. This should inspire you to learn it. I suggest reading Iczelion's essays on Win32 ASM programming. A must read for every cracker. A simple Google search for iczelion should fetch you thousands of links to his site where you can download his tutorials. I have included the source code with this tutorial.

Please go through it. I suggest you make your own program rather than just reading through this tutorial. The files included with this package are

- | | |
|------------------------|--|
| <i>1. musikdlg.asm</i> | <i>The source code file</i> |
| <i>2. rsrc.rc</i> | <i>Resource Script</i> |
| <i>3. mfmpayer.inc</i> | <i>Include file for mfmpayer</i> |
| <i>4. mfmpayer.lib</i> | <i>Library for mfmpayer</i> |
| <i>5. musikdlg.pdf</i> | <i>This tutorial</i> |
| <i>6. make.bat</i> | <i>Dos script file for building the exe file</i> |
| <i>7. music.xm</i> | <i>The music file to be played. This can be replaced by any other xm file with the name music.xm</i> |

Next, copy all the files except musikdlg.asm and musikdlg.pdf to a new directory. Make a new file named musikdlg.asm and follow this tutorial.

Just have a look at all the files. Make.bat is nothing special. Open the file in a text editor and make the necessary changes to the path of MASM32 as per your configuration.

*****Make.bat*****

G:\masm32\bin\rc.exe rsrc.rc

This command runs the resource compiler which compiles the resource script rsrc.rc and produces rsrc.res file which is the resource file

G:\masm32\bin\cvtres.exe /MACHINE:IX86 rsrc.res

This command (convert resource) converts the resource file rsrc.res to object file rsrc.obj.

G:\masm32\bin\ml.exe /Cp /c /coff musikdlg.asm

This command assembles musikdlg.asm to object file musikdlg.obj

G:\masm32\bin\link.exe /SUBSYSTEM:WINDOWS musikdlg.obj rsrc.obj

This command links the files musikdlg.obj and rsrc.obj to executable musikdlg.exe

if exist *.obj del *.obj

```
if exist *.res del *.res
```

These commands delete the temporaries if they exist

Pause

This command pauses the screen (Good for WinXp)

```
*****End Make.bat*****
```

Next open the resource script rsrc.rc in notepad. You should find the following lines:

```
*****rsrc.rc*****
```

```
#include "G:\masm32\include\resource.h"
```

```
#define IDM_MUSIC 200
```

```
IDM_MUSIC    RC_DATA    "music.xml"
```

Let's examine it line by line. The first line includes the file "resource.h" which is a header file which contains definitions of CONSTANTS related to resources. In our case the constant RC_DATA.

The second line is definition of the resource ID of the music file. We define it as any number. 200 was a random number. More about ID's... Every resource has an ID. In a program we refer to a resource by its ID. The resources specified in the resource script are compiled into the resource section of the exe file. This means that the music.xml file will be embedded in the final exe file. Also, we'll refer to music.xml in our program by its ID (200).

The third line is the declaration of the resource. Here IDM_MUSIC is the ID of the resource. (Instead of IDM_MUSIC, we could have used 200).

RC_DATA is the resource type. Here, its just data (array of bytes).

Music.xml is the name of the filename. Quite easy!

That's all for the resource script. I'll move on with the asm file. I won't explain the syntax and other programming details. Iczelion has done a great job on this and I assure you that his tutorials are of great quality. I'll just explain you the part about injecting and playing music.

Mfmplayer.lib is a library file which contains the code for playing music. This library contains a function as defined in the file mfmPlay.inc

mfmPlay PROTO :DWORD

*This function named mfmPlay takes a DWORD parameter which is the **pointer to the music data in memory**. This function requires the first dword pointed by the parameter to be the **size of the data**. If the parameter is the pointer to data, music starts. If it's 0, the music stops. Let's inspect the source code for understanding this better.*

.386

.model flat,stdcall

option casemap:none

*include /masm32/include/windows.inc
include /masm32/include/kernel32.inc
include /masm32/include/user32.inc
includelib /masm32/lib/kernel32.lib
includelib /masm32/lib/user32.lib
include mfmplayer.inc
includelib mfmplayer.lib*

Nothing unusual in this part except the last two lines. The library mfmplayer.lib should be included for the mfmPlay function to work and the include file mfmplayer.inc contains the prototype for the function mfmPlay. As simple as this.

The steps involved in playing the music are as follows.

- 1: Get the handle of the resource using function FindResource*
- 2: Load the resource in memory using LoadResource*
- 3: But our requirement needs the first Dword of the resource to contain the size of the resource.*
- 4: To solve this problem, we'll allocate some memory and write the size of resource to its first DWORD. Next, we'll copy the resource byte by byte to newly allocated memory.*

5: We'll play the music using the function `mfmPlay`

Let's look at the source code for doing these things.

```
.const  
IDM_MUSIC      equ      200
```

This is the code for declaring the constant values used in the program. Here we declare `IDM_MUSIC` as 200. This number should be the same as that defined in the Resource Script. Nothing special in here.

```
.data  
msgtext db "Musik Rocks",0           ;Strings used in the program  
msgcaption db "It's cool",0
```

```
.data?  
hInstance DD ?                       ;Uninitialized data  
hResource DD ?  
hDataOld DD ?  
SizeRes    DD ?  
pData     DD ?
```

Even these are the declarations of initialized and uninitialized variables. Let's have the look at the code. Here we'll examine it line by line.

```
INVOKE GetModuleHandle,NULL
```

The **`GetModuleHandle`** function returns a module handle for the specified module if the file has been mapped into the address space of the calling process. The handle is placed in `eax`.

```
mov hInstance, eax
```

The handle of the executable file is stored in the variable `hInstance`.

```
;Loading the music-----
```

INVOKE FindResource, hInstance, IDM_MUSIC, RT_RCDATA

*The **FindResource** function determines the location of a resource with the specified type and name in the specified module.*

It takes three parameters. The first parameter is the handle of the calling module. Second parameter is the resource name. Third one is Resource Type. Here RT_RCDATA. Take a look at the function in MSDN LIBRARY.

```
HRSRC FindResource(  
    HMODULE hModule, // module handle  
    LPCTSTR lpName,  // resource name  
    LPCTSTR lpType   // resource type  
);
```

*If the function succeeds, the return value is a handle to the specified resource's information block. To obtain a handle to the resource, pass this handle to the **LoadResource** function.*

If the function fails, the return value is NULL.

```
mov hResource, eax
```

This command stores the handle of the resource's information block to hResource.

INVOKE LoadResource, hInstance, hResource

*The **LoadResource** function loads the specified resource into global memory.*

```
HGLOBAL LoadResource(  
    HMODULE hModule, // module handle  
    HRSRC hResInfo  // resource handle  
);
```

If the function succeeds, the return value is a handle to the data associated with the resource.

If the function fails, the return value is NULL.

```
mov hDataOld, eax ;moves the handle to data of music to hDataOld
```

INVOKE SizeofResource, hInstance, hResource

This function returns the size of Resource (The music data in our case)
mov SizeRes,eax ;copies the size of music data to SizeRes

INVOKE LockResource,hDataOld

The LockResource function locks the specified resource in memory.

*LPVOID LockResource(
 HGLOBAL hResData // handle to resource
);*

If the loaded resource is locked, the return value is a pointer to the first byte of the resource; otherwise, it is NULL.

mov esi,eax

Moves the pointer to the first byte of resource to esi

mov eax,SizeRes ;No explanation needed ;-)

add eax,SIZEOF SizeRes ;Increments the size of resource by the Size of the variable SizeRes. The use of this will become clear in the next instruction.

INVOKE GlobalAlloc,GPTR,eax

The function GlobalAlloc is used to allocate memory. The first parameter is allocation attributes. Here it is GPTR which means the return value of this function will be a pointer to the newly allocated memory and all bytes will be initialized to zero. The second parameter is the size of memory to be allocated. Here it is Size of music plus one extra dword to store the size of the music as per the requirement of the function mfmPlay.

mov pData,eax

After this instruction pData will point to the newly allocated memory.

mov ecx,SizeRes

Ecx contains the size of the music data

```
mov dword ptr[eax],ecx
```

The size of the music data is copied at the first dword of the newly allocated memory.

```
mov edi,pData
```

edi points to the newly allocated memory.

```
add edi,SIZEOF SizeRes
```

increments edi by a dword

```
rep movsb
```

This command repeatedly copies the data byte by byte from value pointed by esi to edi until ecx is zero. This command is used to copy music data to newly allocated memory. So now after this call, we'll have a newly allocated memory which contains the size of resource appended by the music data. All our part is over. Now call the function mfmPlay and pass the pointer to the newly allocated memory as parameter.

```
INVOKE mfmPlay,pData
```

This is the most important code which you already know.

```
INVOKE MessageBox,NULL,ADDR msgtext,ADDR msgcaption,MB_OK
```

```
INVOKE mfmPlay,0 ;Stop the music
```

```
INVOKE ExitProcess,NULL ;Exit from the program
```

```
end start
```

I was in a hurry to complete this tutorial. Hope it is informative to some extent.

Queries, Appreciations, suggestions, criticisms (if any! ;-) should be addressed to fornix@fusionrulez.cjb.net

Greets: Blue_Orka, Hutches son, iczelion, Lise_Grim (for mfmplayer.lib)

-Fornix