PunchIt - V1.0 - Technical Abstract

Version 1.0
March 2007

Editor: Condzero

Introduction

The idea seemed simple. Write an application that can enhance virtually any application by adding sampled sound effects, chiptunes or background music and support a wide variety of music formats, be free for public use and simple to use. Throw in a couple of custom bitmaps to add depth to the application's interface and, of course, play some music while the client creates their masterpiece. The application should be non intrusive, run on most windows clients, and most important, not modify any existing data on the client's machine. This abstract represents a compilation of the various key issues, the dialog templates and subsequent coding that were born out of the frustration and pitfalls in designing and making everything come together in harmony.

This application represents the author's best attempt at realizing this vision and producing a commercial grade product. Though I relied upon 3rd party products and adapted other's ideas and code to bridge the gaps and speed up the development process. I have given credit where credit is due and list the references where applicable. There are some interesting side effects from this project that could be useful to some. The techniques used in developing this software could be applied to some of the following:

- Developing a PE bundle application
- How to include music in an application
- Bundling resources into an executable and extracting them to disk
- Adding and realigning new sections to a PE
- Creating an invisible console child process using an anonymous pipe for redirecting standard output and standard error messages
- Creating a separate thread to control a progress dialog
- Writing applications that work on high-density displays by scaling fonts and graphics
- How to incorporate a wrapper program and communicate with it during the build process

Source code for both the main PunchIt program and the wrapper program "punchitbin.exe" have been included in the distribution. There are numerous comments interspersed in the code to help provide clarity. Please refer to the source code while reading this abstract and hopefully things will be much clearer. Also, the references provided are a good source of information and greatly facilitated the development of this project.

Disclaimers

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This tutorial is also free to distribute in its current unaltered form, with all the included supplements.

All the commercial programs and referenced code used within this document and program are subject to their respective terms of use. No distribution of commercial applications has been done under any media or host. ARTeam or the authors of the paper cannot be considered responsible for damages to the companies holding rights on those programs. The scope of this tutorial is to improve one's knowledge and appreciation of software development.

Verification

ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: http://releases.accessroot.com

Table of Contents

1. PunchIt V1.0 – Technical Abstract

1.1. Abstract

This Tutorial will introduce you to the methods and challenges of creating a generic application that can add music to virtually any executable file. The source code is also included in this distribution. Tools used in the development of this program include:

- Lcc-win32 V4.0 (Ref D) "C" compiler/linker, framework and developer ide which also includes a debugger and resource editor (LCC-Win32 is a C compiler system for Windows - if not used commercially it is free)
- ResEdit V1.3.5 (free resource editor) (Ref E)
- Ollydbg v1.10 for debugging / testing
- Bass Audio Library V2.3
- Bitsum Technologies – **PECompact2** advanced Windows executable compressor.
  Student Version 2.78a (final)
- Incompetech.com – providers of royalty-free music (donations appreciated) and much more! (Ref H)
- Corel Paint Shop Pro Photo XI (Trial) for digital imaging (Ref I)

1.2. Key Issues – Design Considerations

- Single vs. Multiple music file selection
- What to use and how best to implement music playback
- Embedding resources in an application
- Whether to use overlay data, update resource API's, or adding and realigning new sections to a PE file
- Compressing the final product
- A workable, scalable dialog template that provides visual appeal and says something about the application
- Stub program vs. Wrapper program
- How to test high-DPI Applications
- The About Box

1.3. Single vs. Multiple Music Files

The decision to allow only a single music file was made partly due to design considerations as well as the abundance of music software available to create mixes and condense several music tracks into a single soundtrack. The "sound loop" feature allows for continuous playback of any music source irregardless of size.

1.4. The BASS Audio Library

I have some experience with using this product and must profess at its sheer power, flexibility and ease of use and is available for immediate download (Ref F) and free for non-commercial use.
The download includes numerous examples in C/C++, Delphi, Masm and Visual Basic plus a help file, the BASS.dll and include lib's for integrating with your application. The base module is small, efficient and integrates seamlessly, yet supports many different music formats. This was a prerequisite. The coding examples and help are well documented so that even a novice programmer could implement music into an application in a very short time frame. The decision on what to use and how best to implement music playback was made easy with this product. The key issues were whether to include the bass.dll as a separate external file in the distribution, or incorporate it as an imbedded resource, how and where to launch it and what functions were necessary in its implementation. BASS

requires DirectX 3 or above for output. BASS does not require that a soundcard with DirectSound/DirectSound3D hardware accelerated drivers is installed, but it does improve performance if there is one. BASS also takes advantage of MMX, which improves the performance of the MOD music playback.

In order to integrate the bass.dll into this application, the following was necessary:

To use BASS with LCC-Win32, you'll first have to create a compatible import library for it. This is done by using the PEDUMP and BUILDLIB tools that come with LCC-Win32. Run these 2 commands:

PEDUMP /EXP BASS.LIB > BASSLCC.EXP
BUILDLIB BASSLCC.EXP BASSLCC.LIB

and then use BASSLCC.LIB in your projects to import BASS:

```
LIBS=c:\lcc\lib\basslcc.lib  c:\lcc\lib\shell32.lib
EXE=punchit.exe
```

The implementation will be based on your language choice.
One of the first decisions was to include the bass.dll as a resource. This has the advantage of controlling which version is used (dll hell) and will be less likely to get lost or deleted accidentally in the course of processing.

The following code demonstrates how this was accomplished. The steps involved are thus:

1. Include the bass header and definition files at the top of the program.
2. Load the resource as a temp file in the client's temporary folder and load the module and all relevant functions used in this program.

```
#define BASSDEF(f) (WINAPI *f)    // define the functions as pointers
#include <bass.h>

// load temp BASS.dll resource file and the required functions
BOOL LoadBASS(void)
{
	rsrcfind=FindResource(NULL, (LPCTSTR)IDR_RCDATA1, RT_RCDATA);
	dwESize=SizeofResource(NULL, rsrcfind);
	rsrcload=LoadResource(NULL, rsrcfind);
	rsrclock=LockResource(rsrcload);

	// get a temporary filename
	GetTempPath(MAX_PATH, temppath);
	GetTempFileName(temppath,"bas",0,tempfile);

	hFile = CreateFile(tempfile,
		GENERIC_WRITE,
		0,
		NULL,
		CREATE_ALWAYS,
		FILE_ATTRIBUTE_TEMPORARY,
	NULL);

	if (hFile == INVALID_HANDLE_VALUE)
	{
		MessageBoxError("CreateFile: Can't write BASS.DLL");
		CloseHandle(hFile);
		return FALSE;
	}

	WriteFile(hFile, rsrclock, dwESize, &dwWritten, NULL);
	SetEndOfFile(hFile);
	CloseHandle(hFile);

	// load the temporary BASS.DLL library
	if(!(basslib=LoadLibrary(tempfile)))
	{
		MessageBoxError("LoadLibrary: Can't load BASS.DLL");
```

```
            return FALSE;
        }

        // "load" all the BASS functions that are to be used
        #define LOADBASSFUNCTION(f) *((void**)&f)=GetProcAddress(basslib,#f)
        LOADBASSFUNCTION(BASS_ErrorGetCode);
        LOADBASSFUNCTION(BASS_Init);
        LOADBASSFUNCTION(BASS_Free);
        LOADBASSFUNCTION(BASS_StreamCreateFile);
        LOADBASSFUNCTION(BASS_MusicLoad);
        LOADBASSFUNCTION(BASS_ChannelPlay);
        LOADBASSFUNCTION(BASS_ChannelPause);
        LOADBASSFUNCTION(BASS_ChannelStop);
        return TRUE;
}
```

The above code demonstrates how you can use LoadLibrary and GetProcAddress to import BASS, instead of using BASS.LIB. It also demonstrates including BASS.DLL in the EXE as a resource, instead of separate as a file. One benefit of the LoadLibrary method is that it allows you to look for the correct BASS version, because you can load and unload BASS.DLL at any time. This also allows those who'd prefer not to have a separate DLL to store it with the program (e.g. in a resource), write it to disk, load it, use it, free it and delete it.

The downside is that you have to manually import each function that you use, with the GetProcAddress function. It has been made a lot simpler to import BASS this way by the use of the BASSDEF #define.

Once bass has been loaded we need to Initialize the default output device.

```
// Initialize default output device
if (!BASS_Init(-1,44100,0,hwndDlg,NULL))
{
        BassError("Can't initialize device");
        return FALSE;
}
```

The code above references the default output device, frequency (output sample rate), no special flags, handle to our dialog window, Class identifier of the object to create (that will be used to initialize DirectSound). Simple and straightforward.

We are ready to stream and play our music. What happens next is entirely up to you. Whether you choose a stream / sample file (i.e. wave, mp3, etc.) or MOD file and whether you play the file from disk or memory can be addressed. The code below demonstrates the use of an imbedded mp3 resource file which is played from a memory buffer. Since the file is quite small, this consumes relatively few resources. Larger music files can be streamed from disk.

```
//From memory buffer
if (!(stream=BASS_StreamCreateFile(TRUE,wavelock,0,dwWSize,BASS_SAMPLE_LOOP)))
{
        BassError("Can't stream the file!");
        return FALSE;
}

if (!BASS_ChannelPlay(stream,FALSE))
{
        BassError("Can't play stream!");
        return FALSE;
}
```

The code above plays our resource file from memory (TRUE = stream the file from memory.). Wavelock refers to our loaded resource file's pointer in memory. We could have reference a filename on disk. The '0' refers to File offset to begin streaming from (only used if mem = FALSE). A DWORD reference (dwWSize) is our resource music file's filesize. BASS_SAMPLE_LOOP will continuously play the music. We obtained the values for wavelock and dwWsize in the following function which finds and loads the resource:

```
// Load mp3 resource file
```

```
void LoadWave(void)
{
	wavefind=FindResource(NULL, (LPCTSTR)IDR_RCDATA2, RT_RCDATA);
	dwWSize=SizeofResource(NULL, wavefind);
	waveload=LoadResource(NULL, wavefind);
	wavelock=LockResource(waveload);
	return;
}
```

When we are done playing our music file (application exit) we invoke:

BASS_ChannelStop(chan); with the handle to the music and … BASS_Free(); which frees all resources used by the output device, including all it's samples, streams, and MOD musics.

And finally free the bass library and any open handles:

```
if (basslib)
{
	FreeLibrary(basslib);
	basslib=0;
}
```

It is important to note that there are many more BASS functions and you are encouraged to experiment and have fun using the BASS Audio Library.

1.5. Embedding Resources

The advantages of embedding resources at design time into an application are obvious. Resources can be uniquely identified, versioned and controlled and be easily accessible by using the familiar resource API's for finding, loading, etc. within a program. I could have opted for a resource dll, but chose to embed them in the main program for easier distribution, all without the client knowing the behind-the-scenes details. The resources in question are the supporting dll's and exe's from 3rd party software vendors (i.e. the Bass audio library and the PECompact2 (Ref G) console application as well as the PunchIt wrapper program). These programs and dll's are fairly small and can be loaded and unloaded quickly. Of course, in doing so, we need to find a place to extract them to and do this in a consistent and reliable way. There are methods available to directly map and load dll type resources directly to memory (similar to what the windows loader program would do, but these methods were not used). A fairly simple and universal method is to write these "temporary" resources to disk and the best location for temporary work space is the client's Temp folder easily accessible through the windows GetTempPath and GetTempFileName API's referenced above.

The larger question, was "How do I include a file or data with my program that I can extract at run-time and do something with it?". Three methods come to mind:

- Bundle them as embedded resources
- Append them as overlay data
- Add them as new sections

MSDN offers some insight and example code to illustrate how to load / update [embed] resources from one executable to another programmatically. The code snippets below are taken from (Ref A) and answer two basic questions:

"How to include the file?" and "Once the file is included, how to extract it and 'use' it?"

1.5.1 Including the file

In this example, the necessary code to take the entire Windows calculator (calc.exe) and add it as a resource to some test .EXE.. The first thing we must do is open calc.exe in read-only mode, and read the entire file into a buffer.

```
HANDLE hFile;
```