David J. Sushil davidjsushil@gmail.com

Playing Sounds with the Windows API and C#

Playing sounds in Visual Studio .NET's managed environment is not a difficult task. It requires accessing an unmanaged Window's multimedia library (winmm.dll), constructing a list of sound flags, and properly utilizing the PlaySound function.

Accessing an Unmanaged DLL

To begin, let's start a new C# project. By default, the project will load with the following .NET namespaces already listed:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

We need to add one more namespace, System.Runtime.InteropServices. This namespace is responsible for allowing managed code to access unmanaged resources. We add the namespace like any other:

```
using System.Runtime.InteropServices;
```

Having done so, we are now able to access winmm.dll, a Windows multimedia library that contains the function we need in order to play sound effects. We need two lines of code in order to properly access our PlaySound function. The first imports the .DLL file, and the second invokes the function so C# can utilize it. The code is as follows:

```
[DllImport("winmm.dll")]
private static extern bool PlaySound( string filename, int module, int
     flags );
```

We should note something of interest here. First, our DllImport statement (native to C#, but illegal in managed code), is written within brackets, and there is no trailing semi-colon. That is not a mistake!

Notice that in our declaration of the PlaySound function, there are three arguments. The first, `filename`, accepts a string containing the path and filename of the sound we want to play. The second, `module`, allows us to play a sound file from an executable. It is of no consequence to this tutorial, so we pass a zero to this argument every time. The third argument, `flags`, accepts an integer value that describes how the audio should be played. Let's look at our flag options in greater detail.

Audio Flags

In order to properly play a sound, we must describe it. The following code doesn't allow for much creativity on our part. In fact, it will be almost identical from one program to the next. It's simply a list of audio descriptions used by SoundPlay.

```
public int SND_SYNC           = 0x0000;
public int SND_ASYNC          = 0x0001;
public int SND_NODEFAULT      = 0x0002;
public int SND_MEMORY         = 0x0004;
public int SND_LOOP           = 0x0008;
public int SND_NOSTOP         = 0x0010;
public int SND_NOWAIT         = 0x00002000;
public int SND_ALIAS          = 0x00010000;
public int SND_ALIAS_ID       = 0x00110000;
public int SND_FILENAME       = 0x00020000;
public int SND_RESOURCE       = 0x00040004;
public int SND_PURGE          = 0x0040;
public int SND_APPLICATION    = 0x0080;
```

A detailed description of these audio flags exists in the MSDN Library at the following address:
msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_playsound.asp

We need only concern ourselves with a few of these flags. Let's first take a look at the difference between `SND_SYNC` and `SND_ASYNC`. By default, SoundPlay plays files synchronously, which means that the function will not return a value until after the sound has finished playing. This means that Windows will have control of our application until the sound terminates. This is not ideal behavior, especially for computer games. We need to play our sound files asynchronously, meaning that SoundPlay will return a value (and control of our application) immediately – not after the sound ceases. This also allows us to stop a sound from playing at any time, which works well for background music or loops.

Next, let's look at the `SND_LOOP` flag. When using this flag, an audio clip will play through in its entirety, at which point it will begin playing again from the beginning. An appropriate use for `SND_LOOP` in computer games is background music.

Playing Sounds

Now, we have enough information about the SoundPlay function to play, loop, and stop our sounds. Let's look at three individual commands and how they work. First:

```
PlaySound(Application.StartupPath + "/test.wav",0,SND_ASYNC);
```

Here, we call our PlaySound function, and pass three arguments. The first argument is the path and filename for a bit of test audio. We pass a 0 to our second argument (again, because we are playing from a file, and not from another resource). Finally, we pass our

SND_ASYNC flag, which indicates that we do not want the function to wait to return a value. This first example will allow us to play a sound one time.

Next, let's examine how to loop a sound:

```
PlaySound(Application.StartupPath + "/test.wav",0,SND_ASYNC |
SND_LOOP);
```

Here, we again pass our test.wav file as our first argument, and a 0 for our second argument. However, we have changed our flags to indicate we want the sound to loop. We do so by sending both the SND_ASYNC and SND_LOOP flags to our function. This will both loop the sound file and force the function to return a value immediately.

Naturally, the question arises, how do we stop audio from playing once it has begun? The solution is fairly simple – we call PlaySound without a file to play, like so:

```
PlaySound(null,0,SND_PURGE);
```

Here, we pass null as our filename, ignore our second argument, and send SND_PURGE as our flag. And the sound stops!