Loading Resources from a DLL to a Visual Basic Application

Michael Park August 28, 1995

Have you ever created an application in Visual Basic that uses a lot of BMP and WAV files? You could load all of the BMP's into each form so that they will be included with the compiled version of your program. Of course the resulting EXE could become very large and any change to these images will require that you re-compile the program. You could just load them from disk. Loading from disk can be slow so animation can be jumpy and your images are not that secure. Forget sound files. I have seen some shareware programs that allow me to bind the sound files in with the executables but have never tried them. You have to just make sure that any program you distribute also includes the WAV files. This can become a tedious project as the number of sound and graphic files increase. So what do you do? You have written that hot VB application that uses 10 WAV files and 20 BMP's. There is a solution. I put all of my BMP's and most of my WAV files into a DLL and then extract them as needed. Loading from the DLL is much faster for the graphics so my animation is smoother and I know if my DLL is there so are all of my additional resources. Need to change a sound or picture? No need to recompile the program, just open the DLL and edit the resource. A resource is what Windows calls things like icons, cursors and bitmaps. Typically, in C programs, these resources are bound into a DLL along with functions to create useful utilities. Does this mean you need to C to use accomplish this? No just follow me as I go step by step through the process.

First off, you are going to need a resource editor. I use Resource Workshop from Borland. It came bundled with my copy of Borland C++. I never bothered with C, but found the Resource Workshop a useful program to put into my VB toolkit. You can buy just the Resource Workshop direct from Borland. Microsoft also has a version called Visual Workbench or something to that effect. I have never used it. Based on that the rest of my conversation will concentrate on Resource Workshop. The Resource Workshop will not create a DLL. It will however let you look inside one and add,delete and change its components. Seeing that you can't create a DLL, lets borrow one. I copied a DLL to temp directory and renamed it to a meaningful name for my project. I then opened it up in Resource Workshop. A dialog box will pop up showing me what resources are currently. Delete all of these resources by selecting the resource and hitting the delete key. You now have an empty DLL waiting for your resources. I would save this empty DLL to be used as a template in the future. Note that Resource Workshop creates a backup copy of the DLL of the original DLL with the extentsion ~DL. So you have a version of rollback, if needed.

Loading images is pretty straight forward. The images must be in a Windows BMP format. Just select FILE from the menu and then select ADD TO PROJECT. This will bring up a dialog box. Change the File Type to BMP from the drop down list box. Now select the file that you wish to include. If you try to trick the program and point to a file type other than a bitmap file type another dialog box will pop up asking you for the Custom Resource Type. We will explore this option in a bit. Assuming you selected a valid image, the program will give the image a default name. You can change this name if you like by choosing RESOURCE menu item and then RENAME. The name the resource has is important so I would suggest renaming the resource to something meaningful.

We have just added a bitmap to our DLL, now how about a sound file. This will require one extra step. Windows does not consider sounds a standard resource so you need to create a custome resource. Just as we added a bitmap we will select ADD TO PROJECT from the FILE menu. This time select USER DEFINED from the drop down list box for File Type. Now point to the valid sound file. A dialog box will appear asking your for the Custom Resource Type. Select the New Type button and name the new type of resource. Remember the name of this new resource. I suggest using something meningful like WAV for SOUND. Just as with the bitmap, the program will give the new resource a default name, and once again I suggest changing that name to something meaningful. The next time you add another sound resource you will see this new resource type in the Custom Resource Type dialog and just select it from there.

That should be all for Resource Workshop. Play around with it. It is a very powerful tool and should be treated as such. Don't open a file open and then save it unless you backed it up. The program will make a backup as earlier mentioned but just give the program the respect it deserves. It is a high level tool.

We now have a DLL that holds all of our resources. In our case, though resources are sound files and bitmap images. We know need to go over to VB and write the code to get the resources from the DLL into our forms were we can use them. Once again we will start with bitmap image because it is the most straight forward. We will be using API calls to achieve our goal so the first thing we need to do is declare those that we need. Put these declarations in the GENERAL section of a VB form or code module. I suggest using a code module and making the functions we create as generic as possible. Then all we need to do is add these modules to our programs as the functionality is needed. The declarations are as follows:

```
Declare Function LoadLibrary Lib "Kernel" (ByVal lpLibFileName As String) As Integer
Declare Function LoadBitmap Lib "User" (ByVal hInstance As Integer, ByVal lpBitmapName As Any) As Integer
Declare Function GetObj Lib "GDI" Alias "GetObject" (ByVal hObject As Integer, ByVal nCount As Integer, lpObject As
Any) As Integer
Declare Function CreateCompatibleDC Lib "GDI" (ByVal hDC As Integer) As Integer
Declare Function SelectObject Lib "GDI" (ByVal hDC As Integer, ByVal hObject As Integer) As Integer
Declare Function BitBlt Lib "GDI" (ByVal hDestDC As Integer, ByVal X As Integer, ByVal Y As Integer, ByVal nWidth As
Integer, ByVal nHeight As Integer, ByVal hSrcDC As Integer, ByVal XSrc As Integer, ByVal YSrc As Integer, ByVal dwRop
As Long) As Integer
Declare Function DeleteDC Lib "GDI" (ByVal hDC As Integer) As Integer
Declare Sub FreeLibrary Lib "Kernel" (ByVal hLibModule As Integer)
Declare Function DeleteObject Lib "GDI" (ByVal hObject As Integer) As Integer

Type BITMAP
    bmType As Integer
    bmWidth As Integer
    bmHeight As Integer
    bmWidthBytes As Integer
    bmPlanes As String * 1
    bmBitsPixel As String * 1
    bmBits As Long
End Type
```

The overall concept is pretty basic. We will tell the program to load the DLL we just created and then tell it we need one of the bitmaps in there. We reference that bitmap with the name we used when we added the resource. We use an API call to create a container for the object, ie a picture control and then pass that image from memory into the picture control. Notice the BITMAP type. We will need this when we actually move the image from memory into the container. We must then clean up after ourselves. I am not going to profess to a long and thourough knowledge of the Windows API and its use in VB. I can't do that and I won't. If you want a better understanding of why and how these API calls do what they do I suggest you look elsewhere. I can tell you that if you use these following lines of code you will get positive results.

Most of the API calls we are using are functions and thus return a value reporting their success with the given activity or assigning a trackable handle. We will need to set up some variables.

```
Dim hLibInst As Integer
Dim hdcMemory As Integer
Dim hLoadBitmap As Integer
Dim hOldBitmap  As Integer
Dim return_val As Integer
Dim bmpInfo As BITMAP

hLibInst = LoadLibrary(app.Path & "\YOUR.dll")
```

This first command make the DLL available. If the program cannot find the DLL and error will occur. This error triggers before your error trapping kicks in so plan for that. The Windows error trap may be enough. I like to keep specific DLL's like this that are not going to be shared by other applications right with the EXE file. Regardless of the methodology you subscribe to you need to point to your DLL and it must be where you pointed!

```
hLoadBitmap = LoadBitmap(hLibInst, “BITMAP_NAME”)
```
With DLL loaded we need to tell it what we are looking for. Remember when we where working in Resource Workshop and I suggested you change the default name of the new resource to something meaningful? Well here is the reason. We pass the function the name of the bitmap we gave it when we loaded it into the DLL. As an example, Resource Workshop would have defaulted the name of the first bitmap resource added as BITMAP_1. So the command would be
```
hLoadBitmap = LoadBitmap(hLibInst, “BITMAP_1”)
```
The hLibInst is the handle given to the DLL when we loaded it in the first line of code. We now have a handle for the bitmap we want to load..

```
return_val = GetObj(hLoadBitmap, Len(bmpInfo), bmpInfo)
```
The GetObj will read the information about this particular bitmap and load the information into the array variable we created using the bitmap type above. In this example we DIM a variable called bmpinfo as type BITMAP. The fields of this array now contain all of the pertinnet information about the desired bitmap.

```
hdcMemory = CreateCompatibleDC(form1.Picture1.hDC)
```
This command will create a holder, so to speak, for the bitmap. Notice we are passing the hDC for a picture control found on form1 for this example. This can be interpreted as establishing the picture control as the place to put the bitmap.

```
hOldBitmap = SelectObject(hdcMemory, hLoadBitmap)
```
So far we have established a handle for the bitmap with LoadBitmap and a handle for where we want the image to go with CreateCompatibleDC. We now select the object telling the function what its handle is as well as the handle for its destination.

```
return_val = BitBlt(form1.Picture1.hDC, 0, 0, bmpInfo.bmWidth, bmpInfo.bmHeight, hdcMemory, 0, 0, SRCCOPY)
```
Now we BitBlt the bitmap into the awaiting control. This API function is useful for several things. Here we are simply transfering the image from one location to the next.

```
return_val = SelectObject(hdcMemory, hOldBitmap)
return_val = DeleteObject(hLoadBitmap)
return_val = DeleteDC(hdcMemory)
FreeLibrary (hLibInst)
```
The four lines above are used to clean up the resources used for getting the bitmap from the DLL to the picture control. That’s it. You have successfully moved an image from a DLL into your VB application. I thought it may be a good idea to put all of the code together so it would be more straightforward to read. Here it is:

```
Dim hLibInst As Integer
Dim hdcMemory As Integer
Dim hLoadBitmap As Integer
Dim hOldBitmap  As Integer
Dim return_val As Integer
Dim bmpInfo As BITMAP

hLibInst = LoadLibrary(app.Path & "\YOUR.dll")
hLoadBitmap = LoadBitmap(hLibInst, “BITMAP_NAME”)
return_val = GetObj(hLoadBitmap, Len(bmpInfo), bmpInfo)
hdcMemory = CreateCompatibleDC(form1.Picture1.hDC)
hOldBitmap = SelectObject(hdcMemory, hLoadBitmap)
return_val = BitBlt(form1.Picture1.hDC, 0, 0, bmpInfo.bmWidth, bmpInfo.bmHeight, hdcMemory, 0, 0, SRCCOPY)
return_val = SelectObject(hdcMemory, hOldBitmap)
return_val = DeleteObject(hLoadBitmap)
return_val = DeleteDC(hdcMemory)
FreeLibrary (hLibInst)
```

Now that you are a pro at moving bitmaps from a DLL to a VB program lets tackle the moving of sound files. Unfortunatley there are some limatations here and I will discuss those as they arise. First, lets establish the API calls we will need for this routine:

```
Declare Function FindResource% Lib "Kernel" (ByVal hInstance%, ByVal lpName$, ByVal lpType As Any)
Declare Function LoadResource% Lib "Kernel" (ByVal hInstance%, ByVal hResInfo%)
Declare Function LockResource& Lib "Kernel" (ByVal hResData%)
Declare Function GlobalUnlock% Lib "Kernel" (ByVal hMem%)
Declare Function sndplaysound Lib "mmsystem" (ByVal filenae As Any, ByVal snd_async As Any) As Integer
Declare Function LoadLibrary Lib "Kernel" (ByVal lpLibFileName As String) As Integer
Declare Sub FreeLibrary Lib "Kernel" (ByVal hLibModule As Integer)
```

Notice that we are repeating the LoadLibrary and FreeLibrary functions so if you are using both the sound and bitmap loads only declare these functions once. Lets go ahead and set up the variables we need for holding the return values from the functions.

```
Dim hloadwav As Integer
Dim hwaveres As Integer
Dim hsnd As Long
Dim hrelease As Integer
Dim return_val As Integer
Dim hLibInst As Integer

hLibInst = LoadLibrary(app.Path & "\YOUR.DLL")
```

Refer to the previous section on loading the LoadLibrary command. It’s the same as before.

```
hwaveres = FindResource(hLibInst, “WAVE_1”, "WAVE")
```

This next command is similiar to the LoadBitmap command we used above. However, we are now using non-standard Windows resources so we need to let the system know this. We use the FindResource command with the handle of the DLL we just loaded. We pass it the name of the new resource along with the name of the Custom Resource Type we created in the Resource Workshop.

```
hloadwav = LoadResource(hLibInst, hwaveres)
```

Once we found it let’s go ahead and load it

```
hsnd = LockResource(hloadwav)
```

And then lock it , or get a pointer to the memory spot we just assigned a handle to..

```
return_val = sndplaysound(hsnd, 4)
```

This is the actual command to play the sound file. Notice that we pass the value 4, you could use the constant file. The 4 tells the function to play the sound file from memory. If you have used this function before then you have probably used either a 1 or 0 in this spot. The 1 signifies asynchronous play while the 0 signifies synchronous play. This means that if you use 0 the WAV file must play all the way through before resuming processing, not to good for animation. The 1 will start playing the sound and return control back to the program. As I noted earlier, there are some limatitions in storing your sound files this way. The major problem is that the only way for playing sound files with option 4 is synchronous play. So using this method is not so good for playing long WAV files. It will work great for short sounds that release back to the system quickly, though.

```
hrelease = GlobalUnlock(hloadwav)
FreeLibrary (hLibInst)
```

Once again we need to clean up after ourselves. Once again lets put all of the code together:

```
Dim hloadwav As Integer
Dim hwaveres As Integer
Dim hsnd As Long
Dim hrelease As Integer
Dim return_val As Integer
Dim hLibInst As Integer

hLibInst = LoadLibrary(app.Path & "\YOUR.DLL")
hwaveres = FindResource(hLibInst, “WAVE_1”, "WAVE")
hloadwav = LoadResource(hLibInst, hwaveres)
hsnd = LockResource(hloadwav)
return_val = sndplaysound(hsnd, 4)
hrelease = GlobalUnlock(hloadwav)
FreeLibrary (hLibInst)
```

Well there it is! A great new tool to add to your VB tool kit. Obviously, you need to play around with it a little. I would suggest some error checking, and if you are going to use sound you should check to see of the user has a sound card or not. That is for another article. Until then I hope this little bit has helped.

Michael Park
957 Johnson Street
Elmira, NY 14901
(607) 733-7529
mpark@servtech.com