Creating Win32 DLL's in Microsoft Visual C++ 6.0

by Dean Pearce

Introduction

Creating DLL (dynamic link libraries) for Windows based applications can seem like a hassle, especially when you are trying to develop them without any prior experience. Microsoft fortunately has helped Windows developers out by adding the functionality to create Win32 DLL's in their Visual Studio 6.0 suite. Creating a DLL in VC++ (Microsoft Visual C++ 6.0) becomes hassle free, and allows learners to discover how writing dynamic libraries allow their programs to be more flexible and powerful since a DLL can be written in any language, if properly referenced. This tutorial will show you step by step how to create a Win32 DLL in VC++ and how to properly use the DLL from VB (Microsoft Visual Basic 6.0).

Step One: Learn Your Environment

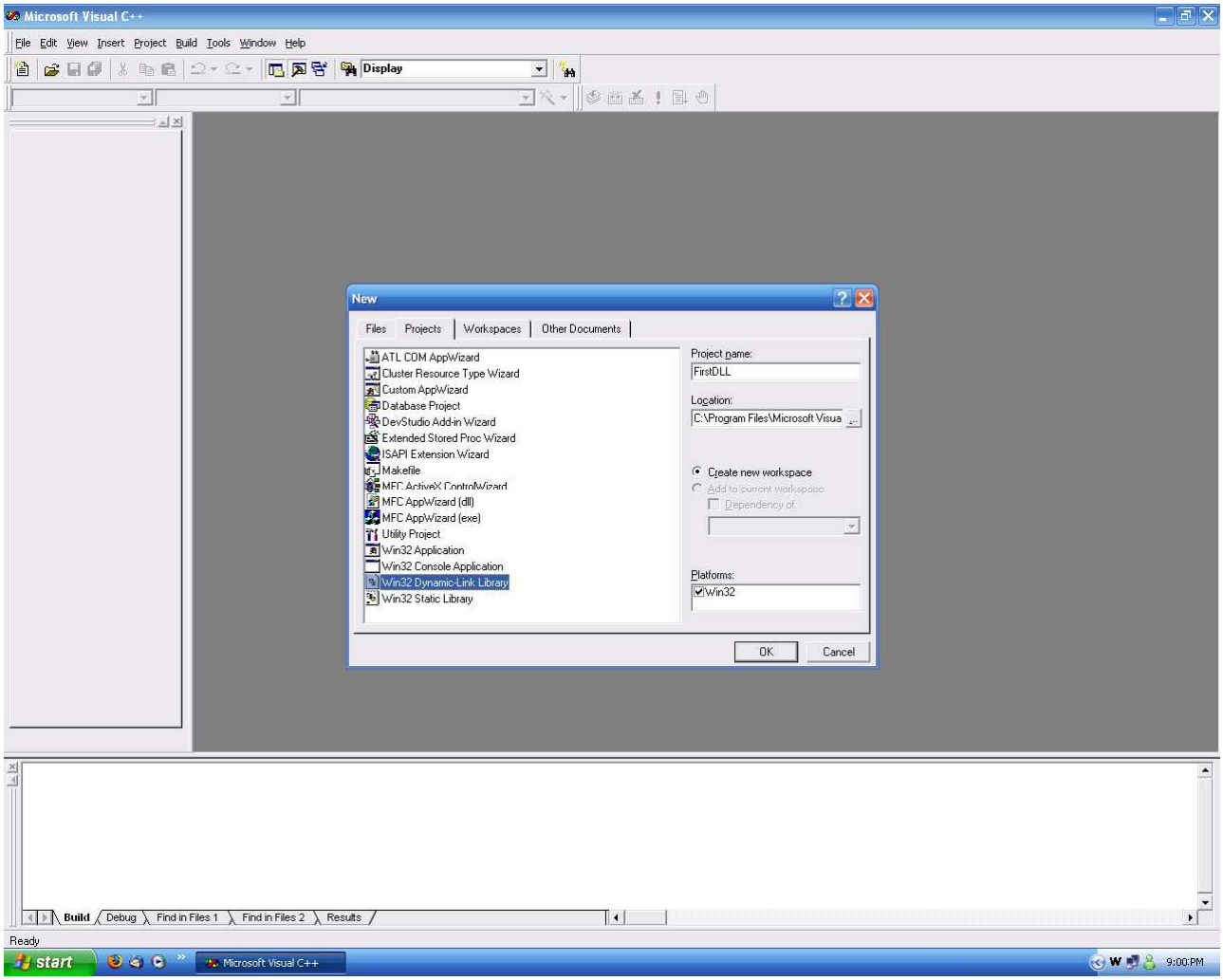
The first step to creating powerful and flexible applications is to know the features of the language you are using and the tools built into your programming environment; for example, Microsoft Visual languages allow for the easy creation of GUI's (graphical user interfaces). For the purposes of the tutorial, I will assume that you have programmed enough GUI based programs in VC++ or VB prior, and are ready to move on to DLL programming.

A DLL is basically a program that has no execution header. Instead the DLL file has what is called an "entry point" of execution, or basically a point where the program can call a function that exists within the DLL. Aside from this fundamental difference, a DLL can be written much like any C or C++ program: you still write functions, return values, manipulate data. Windows 32-bit DLL's have become popular as it allows programmers to write powerful functions in whatever language they please to accompany an executable. It's like organizing your desk: books on one side and papers on the other. The DLL's are like the books, forming a library of functions and the papers are the applications which can reference to the books.

The first thing any programmer should do before taking on any task is to learn the language. If you know why certain things are programmed as they are, it will not be as intimidating when you encounter things like EXPORTS files and _stdcall() functions.

Step Two: Create Your Canvas

Alright, so you are now comfortable enough with the environment and you are ready to undertake the first step in DLL creation. Open up your copy of VC++ (usually found in Programs -> Microsoft Visual Studio 6.0). You will be taken to the main screen. Select File and click New. You will then be presented with the New dialog. Select the Project tab near the top of the window and you will notice a list of possible projects. Near the bottom of the list you should see the “Win32 Dynamic-Link Library” item. Select it and name your project. In this tutorial, I will create the DLL project with the name “FirstDLL.”



The New dialog, with Win32 Dynamic-Link Library Selected and the project named

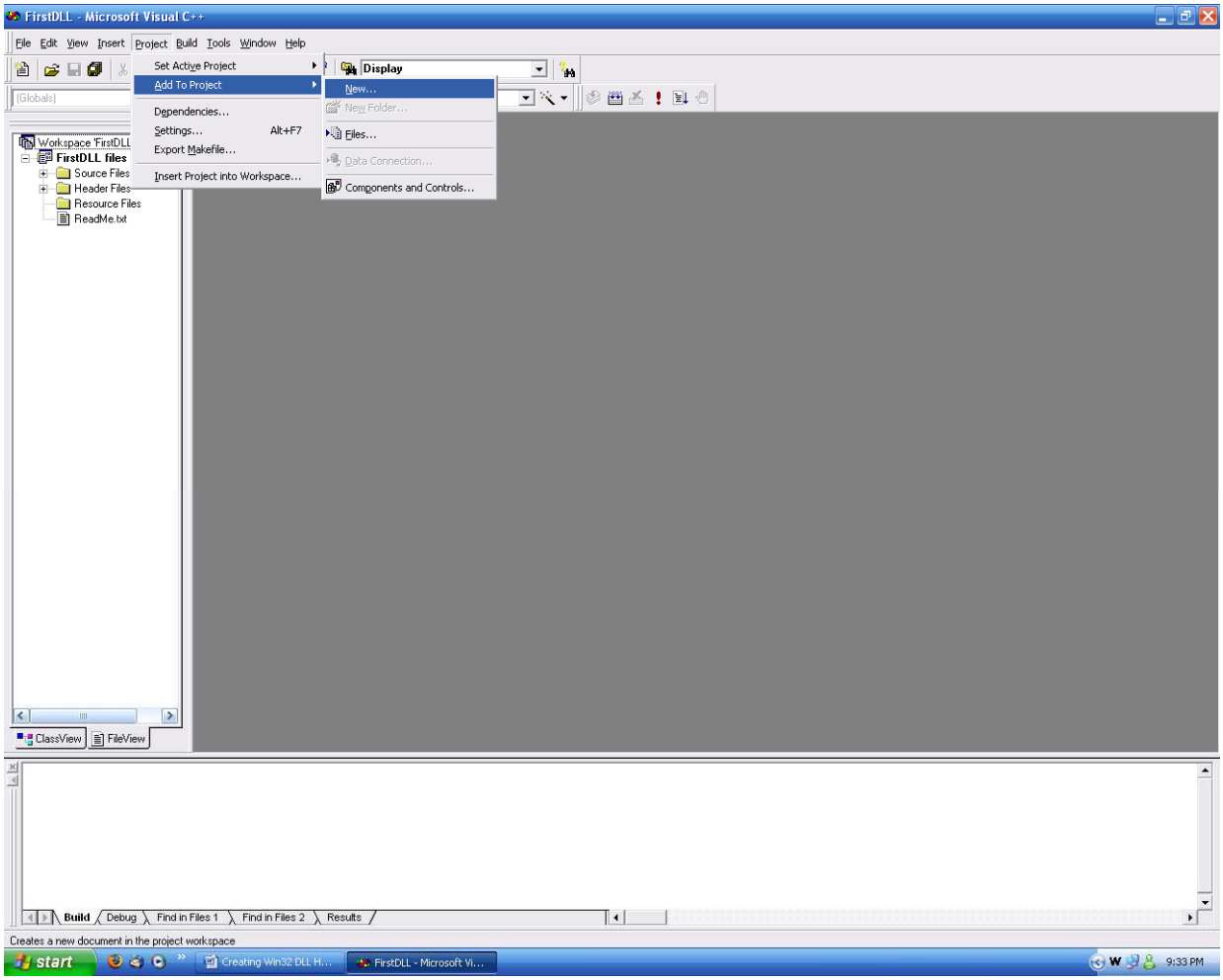
When you are satisfied with the project name and location to store the file, press the OK button and you will be presented with another dialog asking what kind of DLL you want to be created. Select the radial

with “A Simple DLL Project.” After clicking Finish, you will be taken to your project “workspace” (all the files you need to edit and work on your project are automatically opened to make working on projects easier). You’re now one step further into creating your DLL.

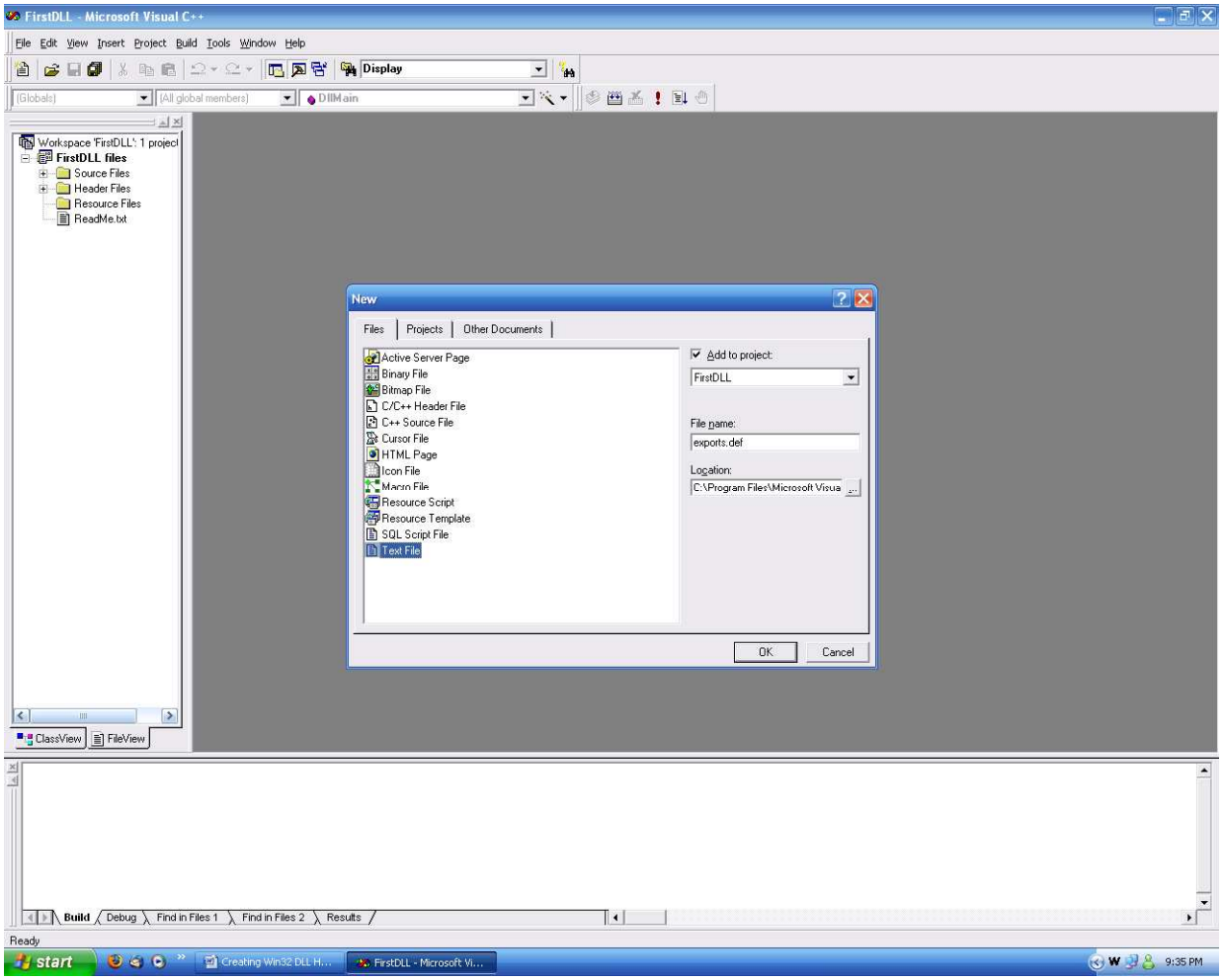
Step Three: Exploring Your DLL and Creating Exports.def

You have now created your DLL, and are well on your way to writing your first working library for use with other applications. However, there is one annoying, but important task to do before your functions are exportable. What does exportable mean you ask? A DLL is essentially a program, and therefore contains functions which complete the various tasks they are designed to do. Even without an exports.def file, you are able to compile your DLL, and in some programming languages you don’t even need the exports.def file, but for maximum flexibility and compatibility you should add one to your project. The exports file is a list of the functions contained within the DLL, and you will have to create it manually with VC++. If you do not include the function in the exports.def file, it would be considered an internal function, and therefore cannot easily be called from another application. The exports.def file is kind of like a menu, allowing an external program an easier time to choose what function it wants the DLL to perform.

To create this powerful and important exports.def, you simply click Project → Add to Project → New. A dialog will pop up, prompting you to select what kind of file you want to add to the project. Select a text file, and in the name field type “exports.def” without the quotes.



Select New from the Add to Project submenu



Select Text File and type exports.def into the File name field.

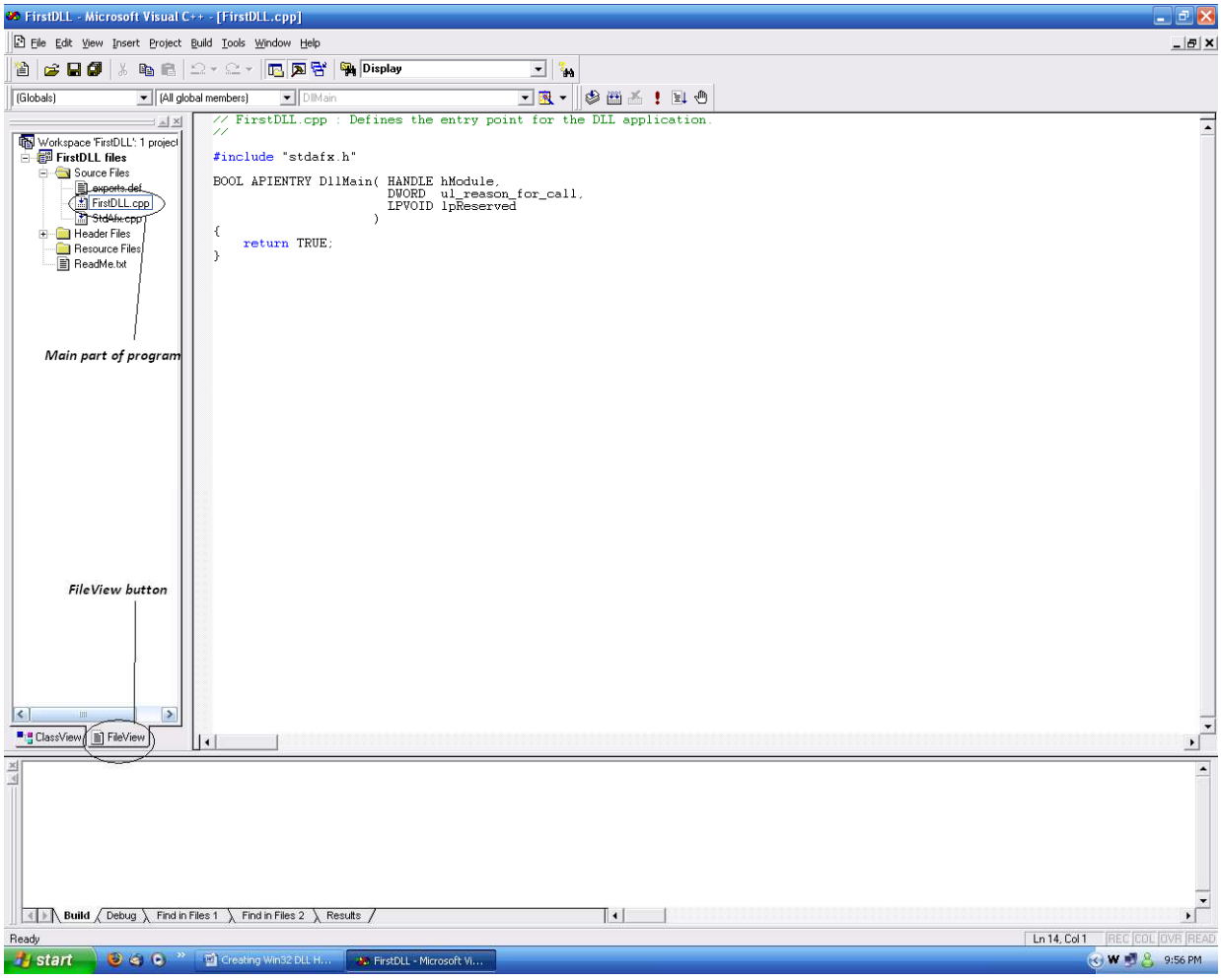
After you have the name entered, proceed to click OK. The file will now be open in the main editing window. Type the word EXPORTS in capital letters, and save your file. Congratulations! You have created the exports.def file. We will be going back to this file, but first we will move on to programming a DLL.

Step Four: Writing code in your DLL

As I stated earlier in the document, I expect by this point you have become at least partially familiar with the one of the languages in the C family of programming languages (C, C++, Objective C). VC++ is flexible in the sense that you can write in the C family language of your choice, as long as you include the correct headers. We'll start by looking at the basics of DLL structure, and how to code a simple DLL. There are more efficient methods and every programmer has his/her own style, but this will be a basic look at how

to code the DLL. We will be building a simple math DLL that can take two arguments and return a completed mathematical operation (+, -, x, /).

On your left “workspace” frame, select “FileView” from the bottom tab. You will then see a “Source Files” tree. Expand it and you will see a file named FirstDLL.cpp (or whatever you named your project with a .cpp extension) double click this file to see your main code window for your DLL.



The main program window in FileView

As you can see in the above window, there is no such thing as the main() function. Instead it has been replaced with DllMain(), which is the entry point for any given DLL. So let’s dive in to some programming. The following block is some sample code that can be used in the project:

```
// Return the sum of two numbers
int _stdcall Sum(int inA, int inB) {
return (inA + inB);
}
```

This piece of code is an example of a function in VC++. I can imagine that either your jaw dropped or you are familiar with the C programming language. This function, named Sum, returns the sum of two integers, inA and inB, which are passed to the function. A function in a DLL returns a value, which the program must interpret. The "int _stdcall" defines what kind of values the function is going to return and how it will return it. The "int" part determines that the function will return an integer and _stdcall is what is aptly called a "standard call," or the universally accepted method to call a function in a DLL. This bit of code is required before the beginning of every function you want to export. Commonly, a DLL's function is to return strings and numerical values, though sometimes the "void" option is used, which return any values, but does internal work in the program.

Once you are working on code within a function, you can follow any programming protocol or standard that you are used to (compiler permitting), as long as you return the correct value for the type of function. A general way to look at the standard function structure for a DLL would be:

```
// Return the typevalue of the function
returntype_stdcall FunctionName(args[]) {

// Code for your function can go here
return (variable_in_correct_return_type);
}
```

Now that you have a simple DLL written, it's time to add your function to the exports.def file which I promised we would be returning to later. From the Source Files Tree on your "workspace" frame open exports.def and add the word "Sum" on the second line. Your file should now contain:

```
EXPORTS

Sum
```

The exports.def file contains a list of function names, as I previously mentioned. As you write functions, it is wise to list their names in this file (one function name per line). When you build your DLL (as we will be doing in the next step) the compiler and linker will automatically make a "menu" of items that an external program can use.

Step Five: Compiling Your DLL

Now that your basic DLL has some code in it, it is time to compile it into a working DLL. Select the Build menu button and select “Build FirstDLL.dll F7” (or whatever your project is called). Now open an explorer window and navigate to your project directory, then into the Debug subdirectory. You will find the DLL that you have built. Since we will want to initially be able to debug the DLL and the program, I recommend that you use the Debug build and not the Release build for most preliminary development.

Now with your compiled DLL in hand, you can start work on a program that will be able to use the DLL. For this tutorial I will be using Microsoft Visual Basic 6.0 (VB), and will base all my declarations around this premise.

Step Six: Using Your New DLL

The VC++ and VB environments are very similar in structure. I will assume that you have used VB prior to this tutorial. Create a new Standard EXE and double click on Form1 to access the Code Explorer. Above the Form_Load() Sub add the following line:

```
Private Declare Function Sum Lib “FirstDLL.dll” (ByVal inA As Long, ByVal inB As Long) As Long
```

This might seem like a complicated line of code, but I will break it down for you. “Private Declare Function” states that you are accessing a function through a private declaration. “Sum” is the name of the function within your DLL “Lib “FirstDLL.dll”” is referencing your DLL library. “(ByVal inA As Long, ByVal inB As Long)” is the type of variables you will be passing to the DLL. And the final “As Long” is the return type of the variable. This might seem like a lot to absorb, but as you learn with Windows API referencing, it is a lot of copy and paste, and simply changing the variable types and function name.

Now, within Private Sub Form_Load() add the following line:

```
Call MsgBox(Sum(2,4), vbOkOnly, “Sum() Function Example”)
```

To break it down, you can call the Function Sum() from anywhere within your code, as long as you pass it the correct variable types, and it will return a value directly into the program. In this example the body

text of the message box will be the sum of 2 and 4. The vbOkOnly and “Sum() Function Example” are simply the style and title of the message box, respectively.

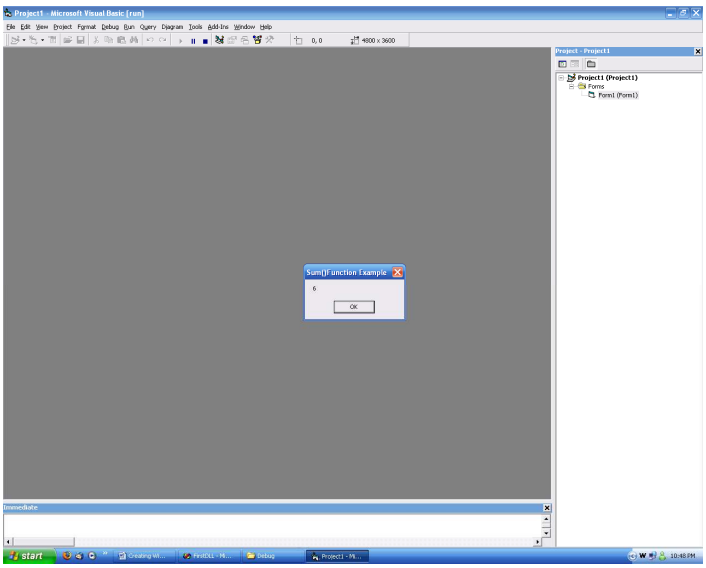
Once you have your code example complete, save and compile your project to an EXE, and save it in a familiarly named folder. Once you have your finished EXE in that folder, copy the DLL from your Debug folder that you opened previously into the same folder as the EXE, and run your program.

A DLL file can be stored in any directory you want, as long as the path is included in the function declaration, but by default, Windows will look in the following directories (not in this order) for the DLL:

- Program’s execution directory
- C:\
- C:\Windows\
- C:\Windows\System32\

It is important to remember that you place your DLL in one of those directories or statically link them from within the program.

If everything compiled as expected, you should see the following message box appear on the screen:



Successfully run example program with a DLL

Congratulations! You have overcome the odds, and have completed a functional DLL that has a list of exported functions that you used through Visual Basic 6.0 as a proof of concept. Mind you, this is only the beginning. Anything is possible once you know how to handle variables properly and return values effectively.

Conclusion

So in summary, VC++ is a very effective and efficient way to create Win32 friendly DLL's. VC++ has a lot of power and flexibility to offer (for example, we did not even touch on MFC DLL's), while keeping things as simple or complicated as the programmer wants them. Included with this tutorial are the sample project files (in Zip® format) so you can browse through the code and learn how to program the DLL hands on. If you have any questions, do not hesitate to contact me at deanpearce@gmail.com, or browse the wonderful tutorials available on the Internet for more assistance. Happy coding!