02.1201.SDI1 :: SDI with a Dialog :: Phase 1

.1) Your First SDI Program in the Visual C++.NET Environment ::

In the previous lab(s), you discovered how to create a "Dialog Based Windows Application". In this lab, we will discover how to create a simple "Single Document Interface (SDI) Windows Application" The rest of this course will focus on working with these types of applications. The third type of Windows application, MDI ("Multiple Document Application") is just the next level up from SDI.

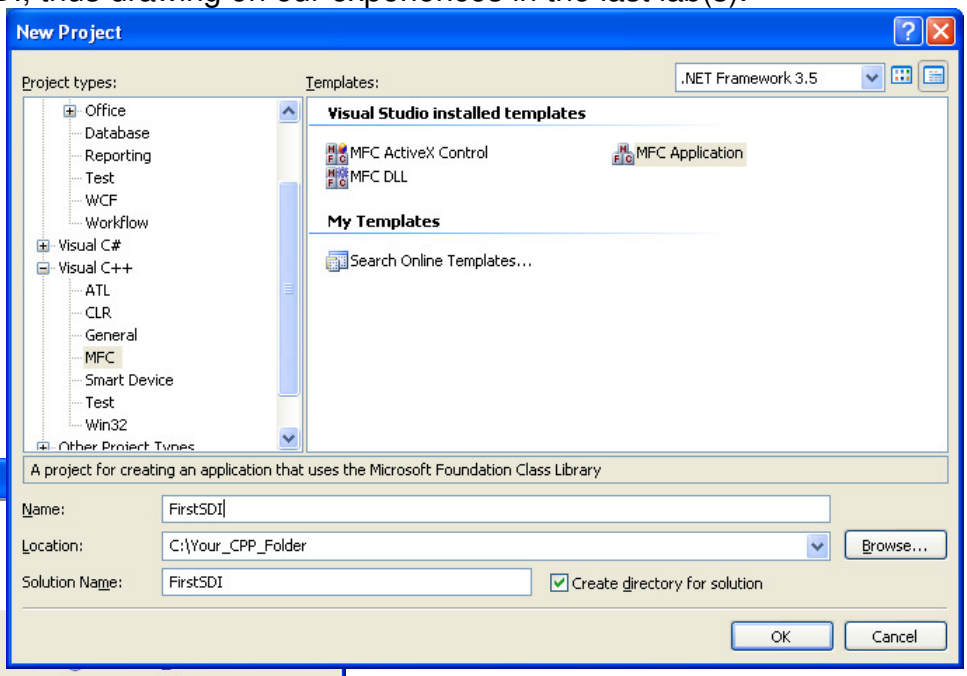
Placing content onto an SDI is more complicated in C++ than you realize. This phase will explore how to "pop up" a Dialog from an SDI, thus drawing on our experiences in the last lab(s).

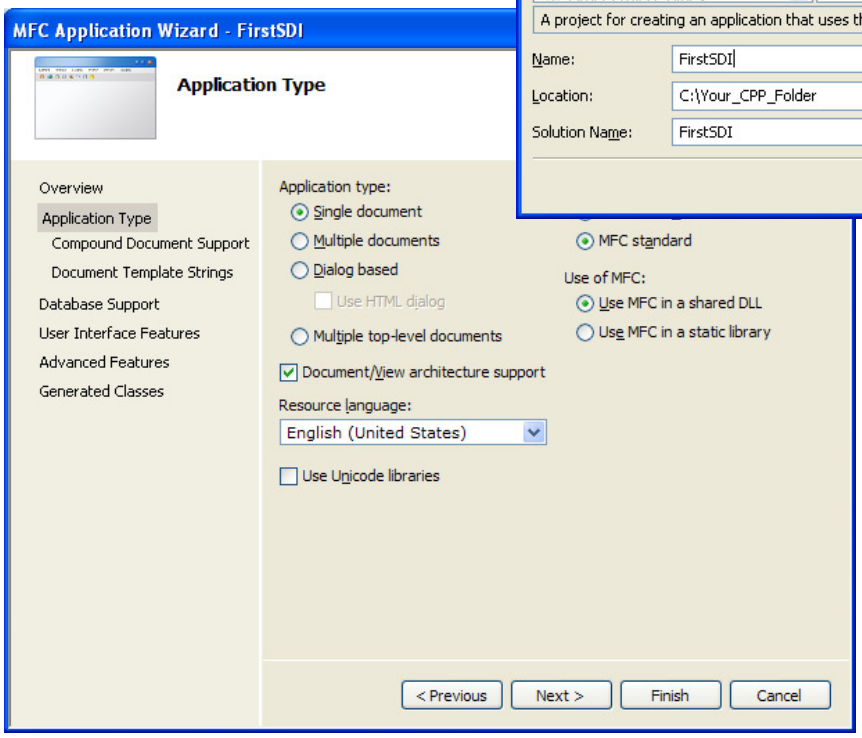
Run Microsoft Visual C++.NET

File... New... Project... Visual C++ Projects... MFC Application

Set ***Name*** to "FirstSDI"

Change ***Location*** to an appropriate spot. Click ***OK***.





Make sure you select

Application Type... Single Document

Use MFC in a shared DLL

Unclick ***Use Unicode libraries***

click ***Finish***

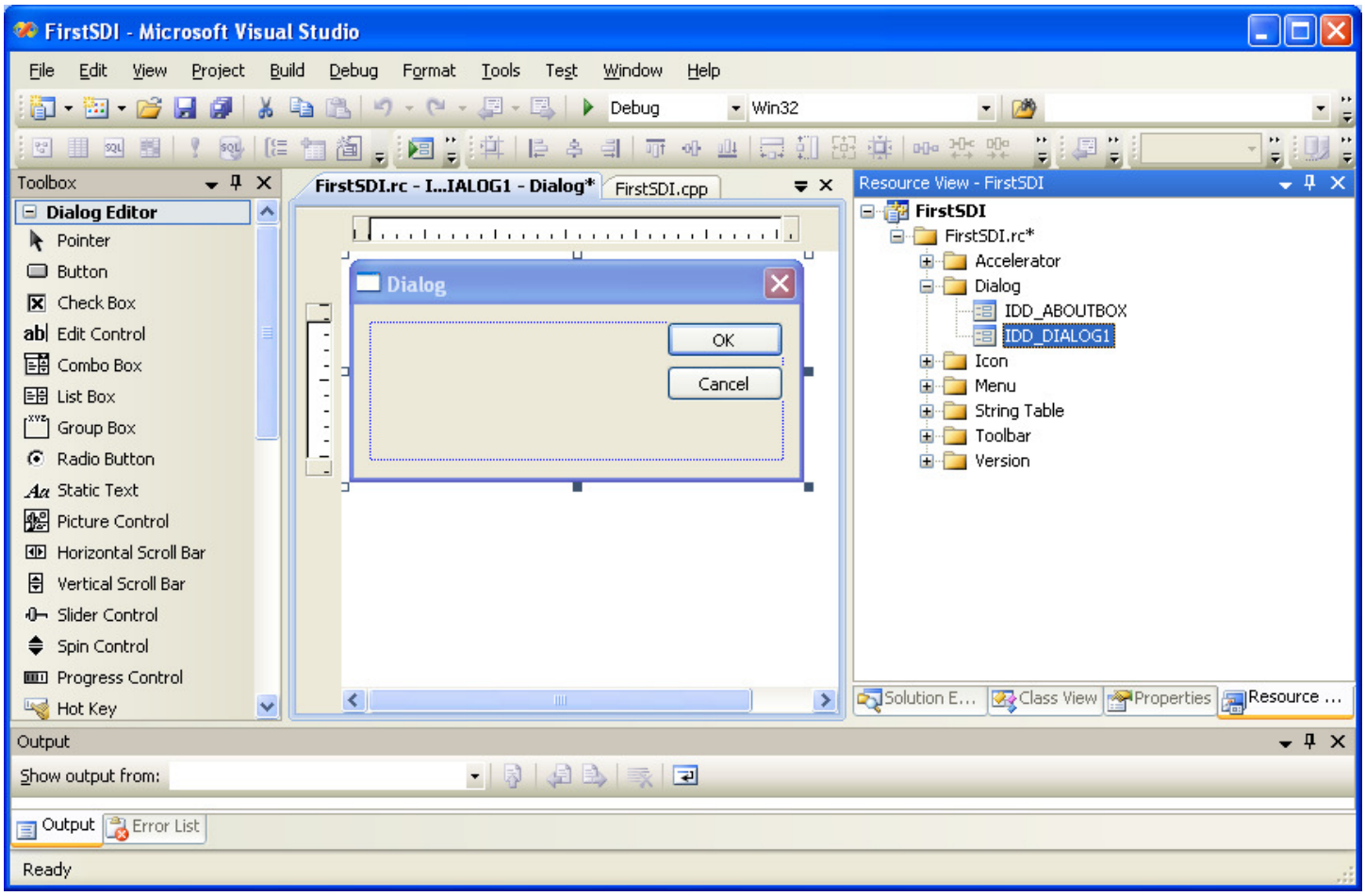
Now, ***Build... Build Solution ,*** and then ***Debug... Start .*** If you don't have any errors, you should soon have your first Visual C++.NET SDI Windows program completed! Hurray!

OK, maybe you're not impressed. <u>BUT, NOW TAKE THE TIME</u> to explore everything that has been generated to accomplish that! Look at everything under ***Solution Explorer***, ***ClassView***, and ***ResourceView***.

.2) Adding a new Dialog Resource ::

In the previous lab(s), the entire application was a Dialog. In this application, however, the application is SDI (Single Document Interface). We will be attaching a “Dialog” to it as a “Resource”.

In your ***Resource View***, ***Insert Dialog*** by right-clicking on the ***Dialog*** folder.



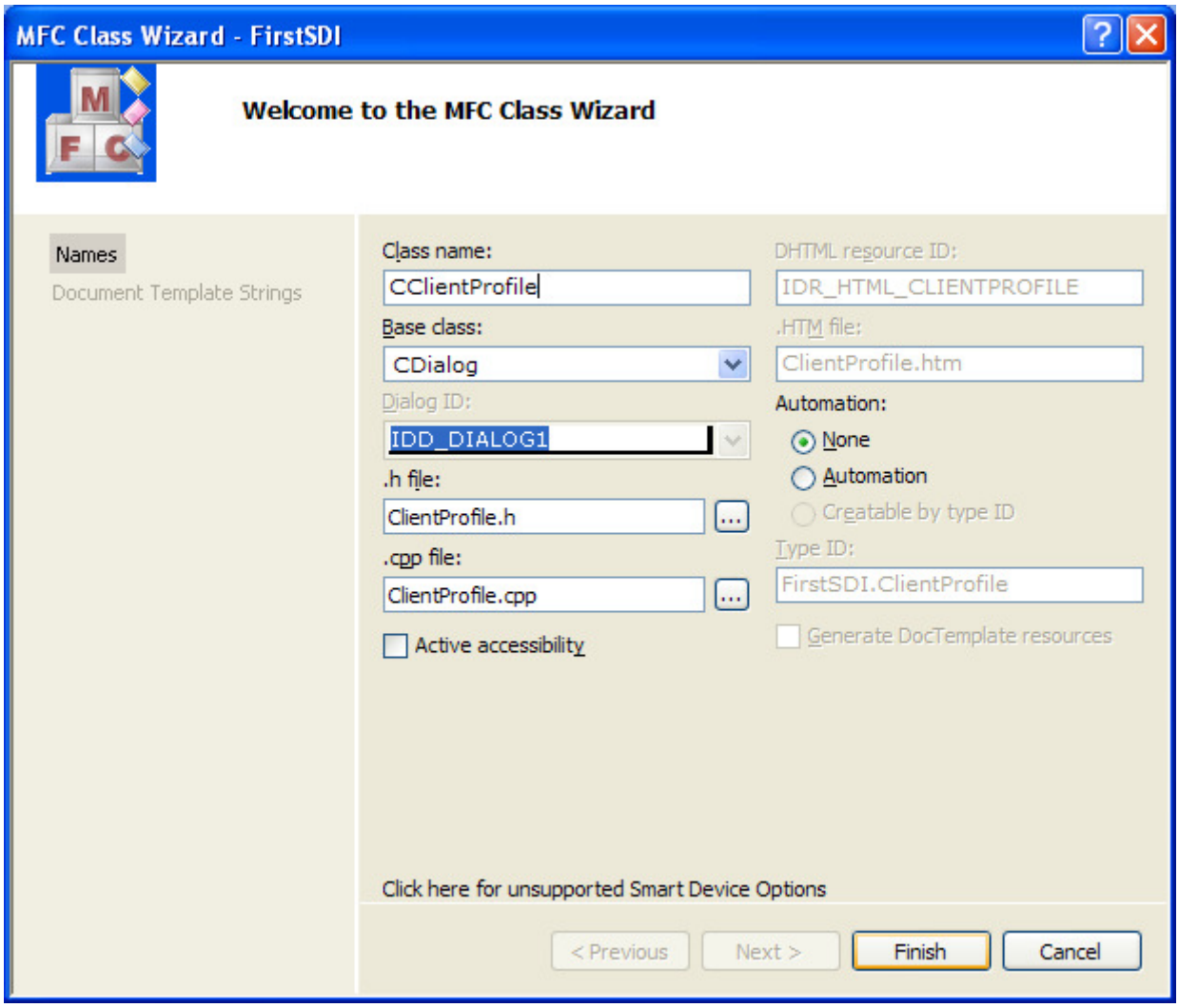
Build, and run your program. It should work, but appear to do “nothing”. That’s because although we have created a new “Dialog Resource”, nowhere in our SDI application have we actually used it!

So far, using advanced tools and wizards, all you’ve accomplish is to draw a pretty picture that looks like a dialog box.

.3) Creating a Class for your New Dialog Resource ::

You have a pretty picture resembling a dialog box. In order for your application to work with it, you need to generate a class for this new resource. You actually did this in a previous lab, but just for review…

Right click on your new dialog box, select ***Add Class*** , and complete the options as follows…

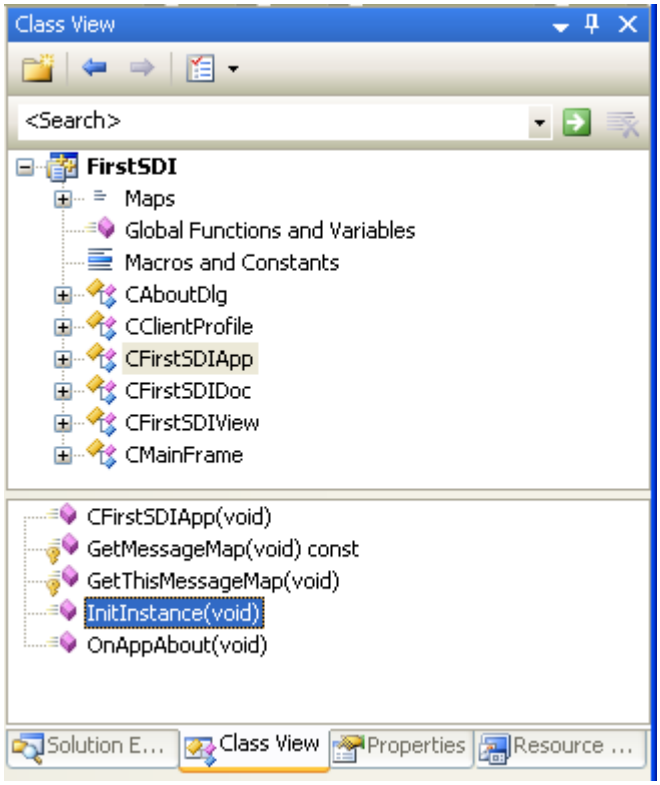


Now, when you look at Class View, you should see your new class has been added to the list.

Also, in Solution Explorer, you should see a new ".cpp" file which will contain code specific to this resource, and a new ".h" file where the Class Definition(s) will be found for this resource.

.4) Calling your New Dialog Resource ::

In your ***Class View***, locate the ***InitInstance*** method for ***CFirstSDIAPP***. , and right click on it.



You should now be in the method that does lots of strange stuff when the application is initialized.

Feel free to look at the code, feel free to be mystified by it. We cannot possibly take the time to explore every single piece of functionality in a Windows program, no more than you understand how the entire engine functions in your car.

What we are interested in doing however, is learning how to "bolt new things on" to this particular engine, and getting additional results.

At the END of ***InitInstance*** method, but BEFORE the return statement, enter this code…

```
CClientProfile dlg;

if ( dlg.DoModal() == IDOK ) {
      AfxMessageBox("You clicked OK");
}
else {
      AfxMessageBox("You clicked CANCEL");
}
```

Take a moment to understand *this* particular code. We are declaring an instance of the ***CClientProfile*** class you created in the previous step, and calling it "dlg". Got that? "dlg" is the variable name of an "instance" of the ***CClientProfile class***.

Declaring a variable "does nothing" except allocate memory. You should know this from your introductory C++ course(s). In this case of this code, we want to call the ***DoModal*** method for the dlg instance. Do you see that?

DoModal passes control over to "all the code" (that we have not written yet) that will be responsible for functionality within this dialog box. It's when you click on "OK" or "Cancel" that control is returned to this particular place in the program. Do you see where I'm capturing the "return value", and invoking logic accordingly?

.5) The Importance of "Including" Class Definitions ::

Attempt to build your program. You should encounter an error message when attempting to declare "dlg" as an instance of CClientProfile. Why????

Because, this cpp files does not know what a "CClientProfile" datatype is. That class definition is found in the file ClientProfile.h . You must INCLUDE this class definition in order to use it.

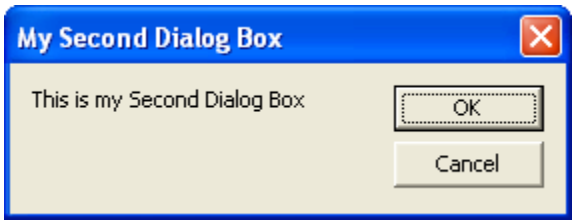
Add this line to the top of the FirstSDI.cpp file, just after the "other" includes…

#include "ClientProfile.h"

Now, attempt to build and run your program. You should now have core functionality, and meet the expectations discussed in the previous step.

.6) Additional Practice ::

Create *another" dialog resource that contains nothing more than a simple message. Its class name should be "CSecondDialog". Also make sure the "Caption" for this new dialog is set to something other than the default… note how I made mine "My Second Dialog Box".



Write the code that "pops up" *this Dialog* AFTER the one we did previously, and display some words of wit after OK or Cancel has been clicked.

.Z) Checkpoint ::

You should know how to generate an SDI Windows Application, and add a new Dialog resource to it. You should be able to create an instance of that new Dialog resource, display it on the screen, and capture whether OK or Cancel was clicked.

There is nothing to submit at this point. You should only ensure you understand the concepts covered to this point.