Creating an IADS Custom ActiveX Control Using C++ VS2005

March 2014

SYMVIONICS Document SSD-IADS-044

SYMVIONICS, Inc.
Telemetry Systems

Created February 4, 2010

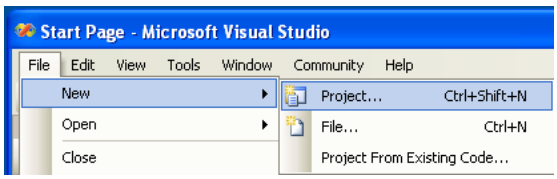
Table of Contents

1. Introduction

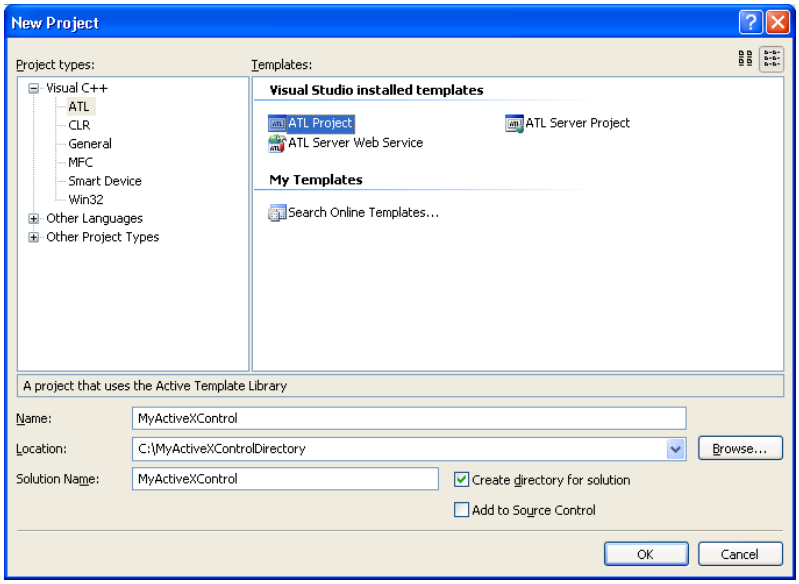
This document assumes you are using Microsoft Visual Studio 2005. The tutorial has not yet been attempted on a newer version, although it may still apply. This instruction guide will cover: creating a new display using the ATL COM Wizard, adding the new control to IADS, debugging the new control in IADS, and ensuring the new display is saved within IADS.

2. Creating Your Display using the ATL COM Wizard

1) Open up VS2005 and Select "File -> New->Project"



2) In the New Project dialog that appears, choose the "Visual C++->ATL" tier and click the ATL Project" option. At this point, please read the next step before you finish completing the dialog. There are some important considerations when choosing the proper project name.

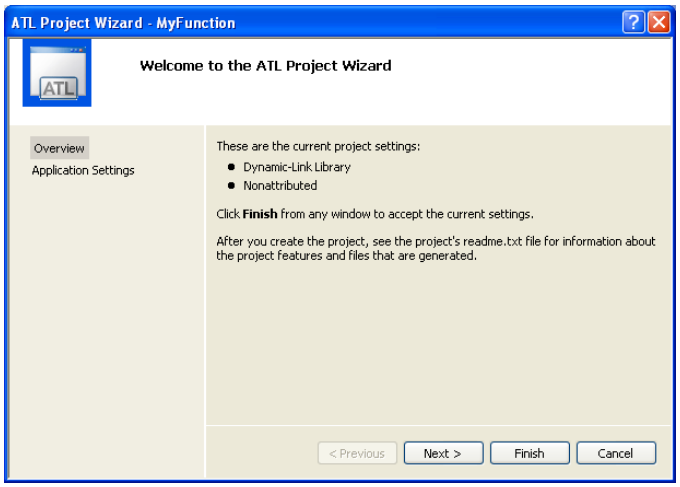


3) The project name you choose will become part of the display identifier name (aka ProgID, see note below). When it comes time to use your control in IADS, users will insert your new control into the "Display Builder" toolbox based solely upon its name (more on this later). Plan on creating many displays in one "project" (most common and easier to manage the code). Choose a general project name like "AircraftGauges" or "FluidSystemDisplays". One way to look at it is that the project name is akin to the "Genus" of your display, so shoot for generality. Consider prefixing the project name with your organization like "Nasa" or "Lockheed", as it may easier for users to locate your control the "Display Builder" list (i.e. NasaFluidSystemDisplays or LockheedAircraftGauges).

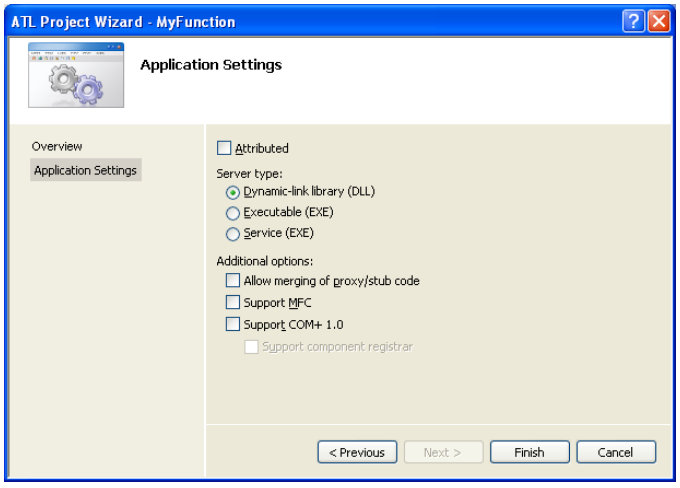
Note: Microsoft refers to your function's name as its "ProgID" (aka Program ID). This is the string equivalent of your GUID (Global Unique Identifier) for the function. These Ids are placed in the Microsoft registry (directly from your project's ".rgs" file), allowing your object to be created without any knowledge of the location of your "Dll" on the file system. Of course, this assumes that it is registered using the "regsvr32" program (consult the Microsoft documentation).

Now, in the fields at the bottom of the dialog, enter the project name, location, and the solution name.

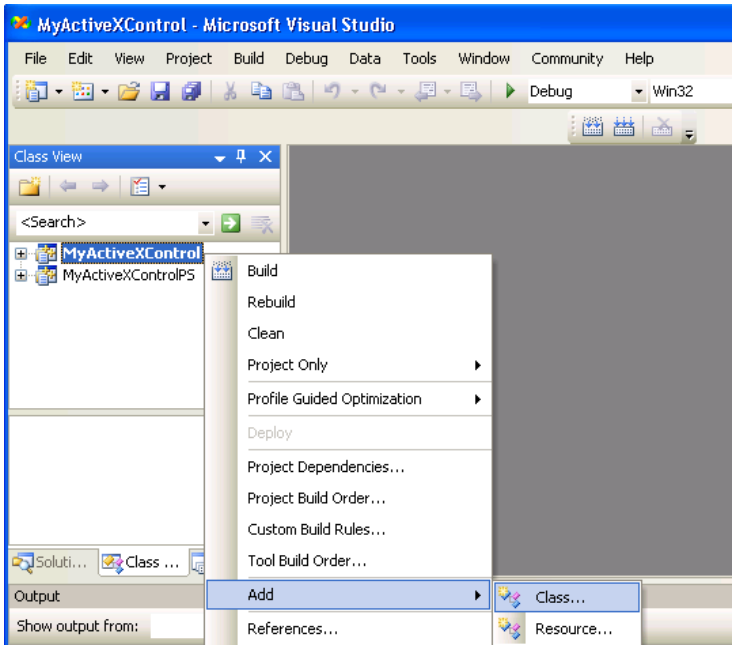
4) After pressing OK, the "ATL Project Wizard" dialog will appear as below.



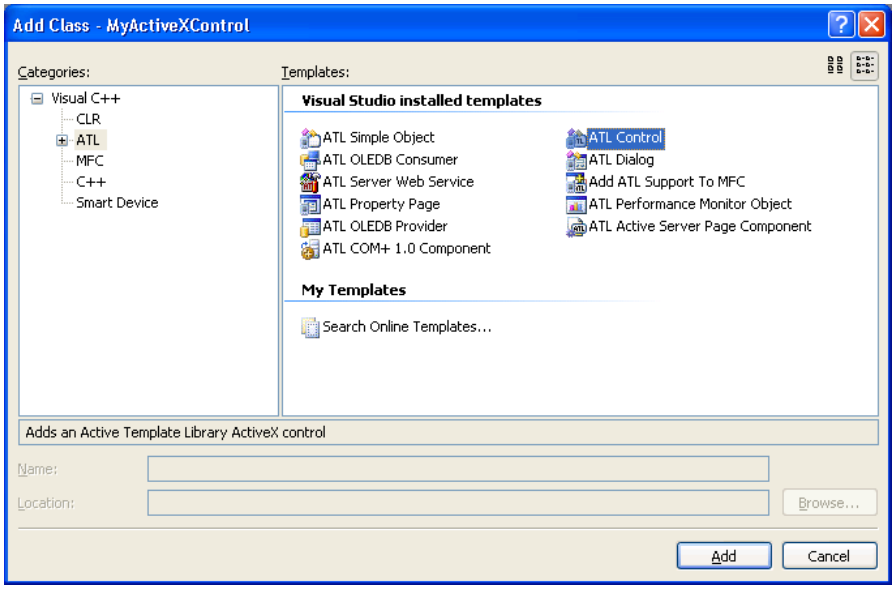
5) Click the Next button in the Wizard. On the new wizard page, ensure that the "Dynamic Link Library (DLL)" is checked. Every display that runs in IADS is of type DLL because it allows for maximum speed in displaying graphics. Press the "Finish" button and the Wizard will set up your project.



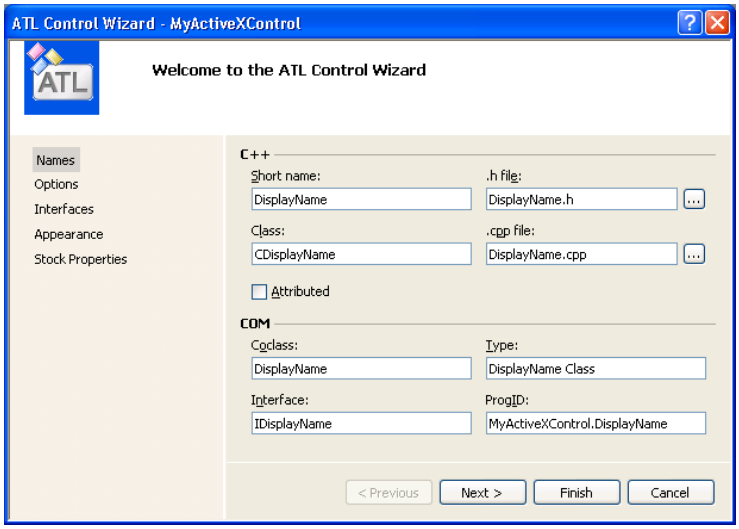
6) Next, go to the “ClassView” tab in Visual Studio’s workspace and right-click on the project name. Choose “Add->Class”.



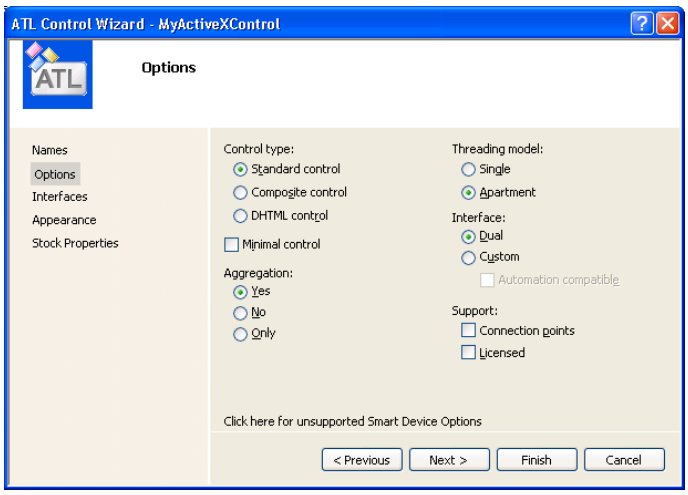
7) Upon adding a new class you will be presented with a dialog. Click the “ATL” tier and “ATL Control” as shown below. When that is complete, press the “Add” button.



8) On the first tab, enter the name of your display in the “Short Name” field. The wizard will fill out the rest of the tab automatically. For this example, I used “DisplayName” as the short name. The name entered will be combined with your project name and will present the final display name inside of Iads (ProjectName.FunctionName) as explained on page 1. See the “ProgID” field in your dialog for your final Iads display name. Update: Newer VisualStudio versions do not automatically populate the ProgID field. Please ensure the ProgID field contains your specific ProjectName.FunctionName text. If not, please type in the appropriate text manually. Press “Next” to continue.



9) On the next tab (“Options”), leave everything as default (Standard control, Apartment, Dual, Yes, and no other options checked). This will allow you to take full control of a “blank canvas” and draw your display using low level graphics libraries such as GDI/GDI+ or OpenGL. On the other hand, if you need to create a “dialog based” display containing typical dialog elements such as text boxes, drop down lists, etc you’ll need to select the “Composite control” choice.

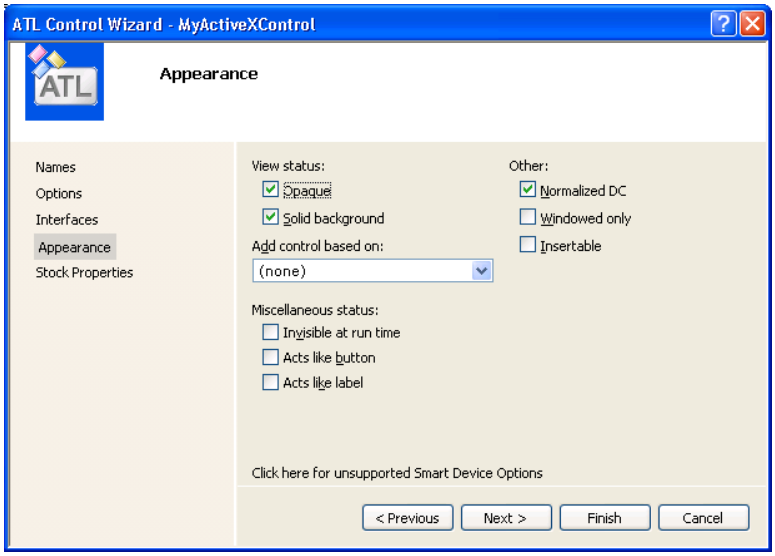


The remaining options are basically “COM speak”. If want understand these options fully, you’ll have to consult the Microsoft documentation. The most notable remaining option is

"Interface". In order to create a real compliant ActiveX "display", you must choose "Dual" interface. This will enable IADS (and other programs) to interface to your control using the "IDispatch" interface, which allows a loosely coupled, "on the fly" communication. This also happens to be the primary (simplistic) way that IADS gets data to your control. More on this subject later.

10) On the next tab ("Interfaces"), leave all of the default choices and select "Next". Again, these options are more "COM speak" and include standard interfaces in which the Wizard will implement for you automatically. If you want more background information, consult the Microsoft documentation.

11) On the next tab ("Appearance"), select the "Windowed Only" checkbox if you plan on using OpenGL; otherwise uncheck it. "Windowed Only" will ensure that we have a window to create an OpenGL context upon. For GDI based displays, we want to attempt to draw "without a window" for speed and resource considerations. Leave the other settings as default (later discuss the speed benefits of de-selecting the "Normalize DC" checkbox). Remember, OpenGL = "Windowed Only". Don't worry, this can be easily changed later if you make a mistake (as can almost anything).

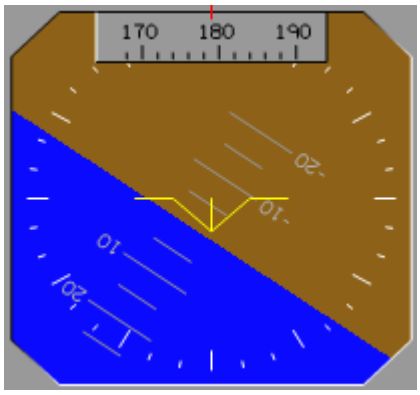


12) On the next tab ("Stock Properties"), leave all the options empty and select "Next". These options are display properties that the Wizard will implement for you automatically. Generally, each property will be added after Wizard is complete. If you want more background information, consult the Microsoft documentation.

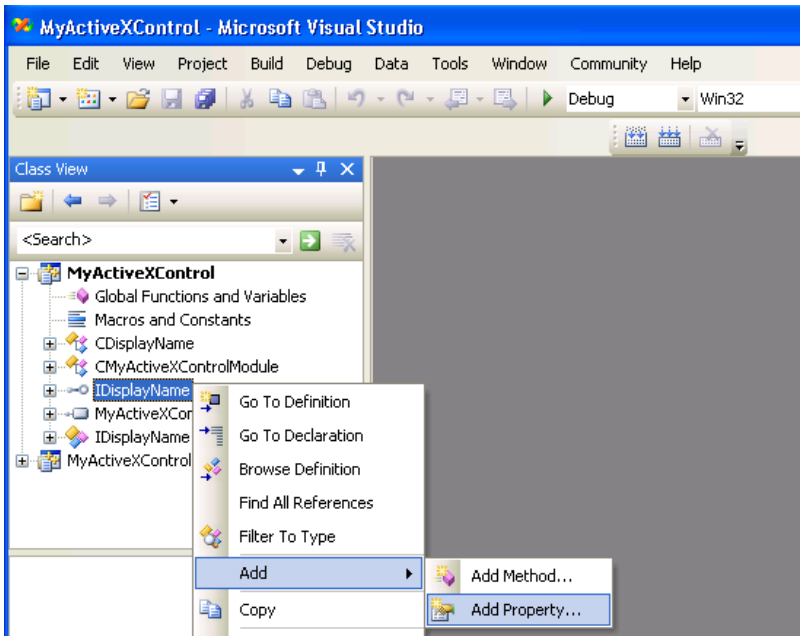
After clicking "Finish", the Wizard will auto-create most of the code needed for your new display. Examine your "Solution Explorer" view… It should now contain the new display object by name. You're half way home now.

2.1. Adding Properties to the Display

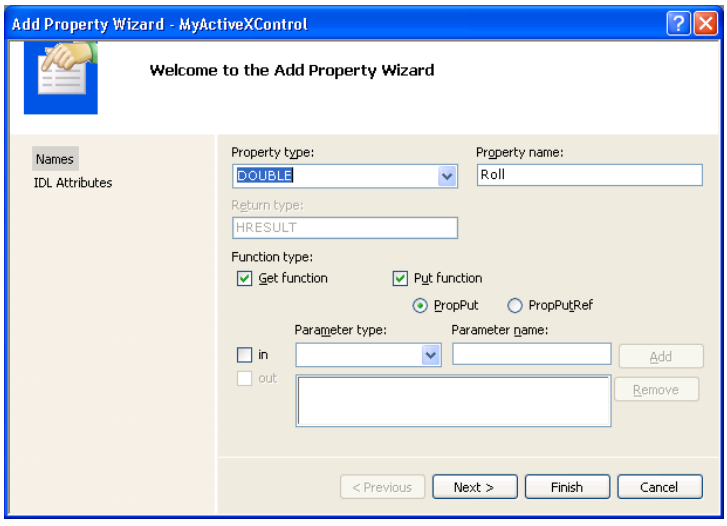
Now it's time to add "Properties". Think of properties as "data injection ports" or "interface plugs". They are really just any attributes of your display (such as text color, needle angle, or scale factor) that you want the user to be able to change or animate. To give a concrete example, the included demo project is code for an "Attitude Indicator" that simulates an aircraft dial. It has properties for "Roll", "Pitch", and "Heading" as well as "SkyColor" and "GroundColor". Any property that you include in your display will be an "access point" on which the user can modify its contents/characteristics/behavior. Changing the "Pitch" property in my attitude indicator example would, as expected, cause the display to rotate its graphics to indicate the new pitch angle. The magic of building an ActiveX control is then to understand the "scope" of your control's behavior, and to provide your users every property that you foresee them changing (within reason, don't go overboard); and also to supply code that responds to these property values and outputs the appropriate response (i.e. draw attitude indicator display at the current value of the roll, pitch, heading, skycolor and groundcolor properties). When this is complete, the user can drive any of these properties, with data from IADS, simply by dragging/dropping a parameter on your newly created display; or they can set any of these properties to a constant value using the "right-click" properties sheet of the display. The best part is that all you have to do is worry about **what** properties to add and how to implement them, and IADS will take care of **ALL** of the data related issues.



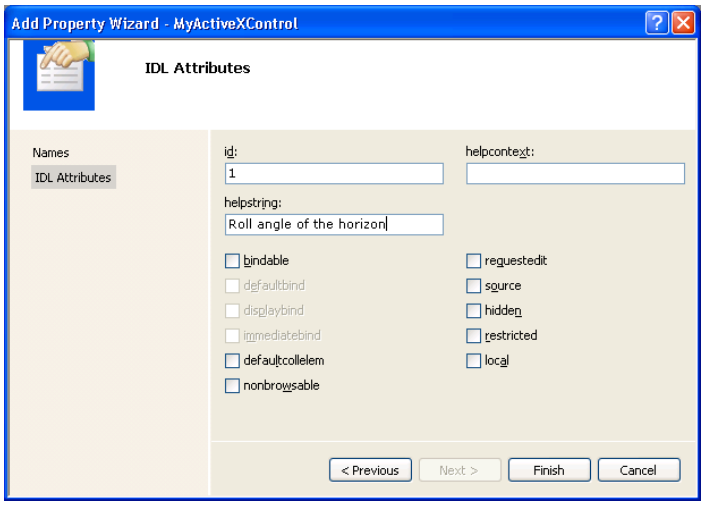
1) Now, make sure that you are in the “ClassView” tab of the VS2005 workspace. To add a “Property” to your new control, right click on the “IXXXXX” where “XXXXX” is the name of your newly created display (look for the little “magnifying glass” icon**)**. Select “Add Property” from the popup menu.



2) In the Add Property Wizard, set the property type to the desired data type (double in this example) and the name of the property (Roll in this example) and click “Next”.

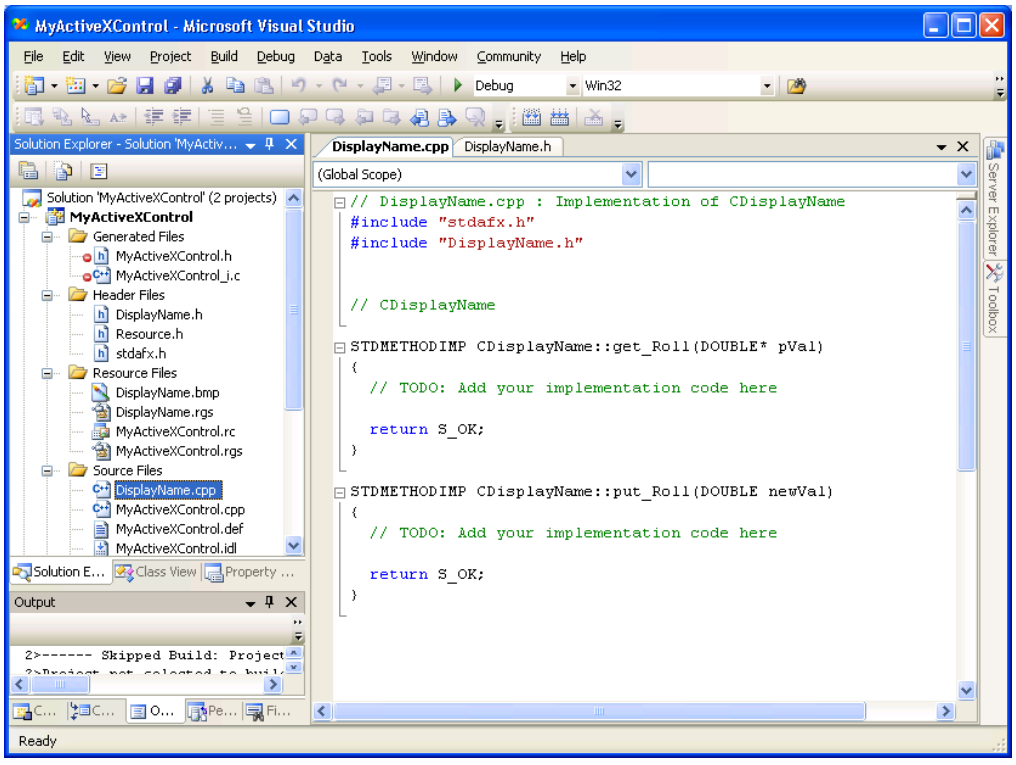


3) In the last page of the Add Property Wizard, leave all the options as default except the helpstring field. This helpstring will be displayed in the Iads properties sheet when the user is setting the property, so try to provide a descriptive (but short) sentence for your new property.



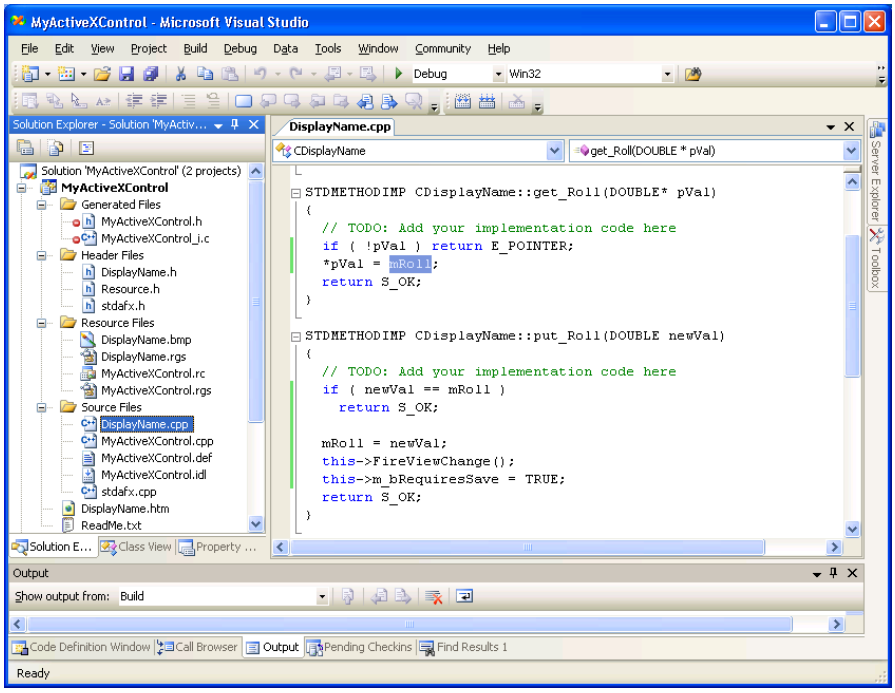
4) When you are complete, press the “Finish” button. This will auto-create code to implement a property named “Roll” within your new project. Repeat this process (starting from step 1) for every property that you want to add to the display.

5) Going back to your “Solution Explorer”, you’ll see the new property code inserted into your display’s .cpp file.

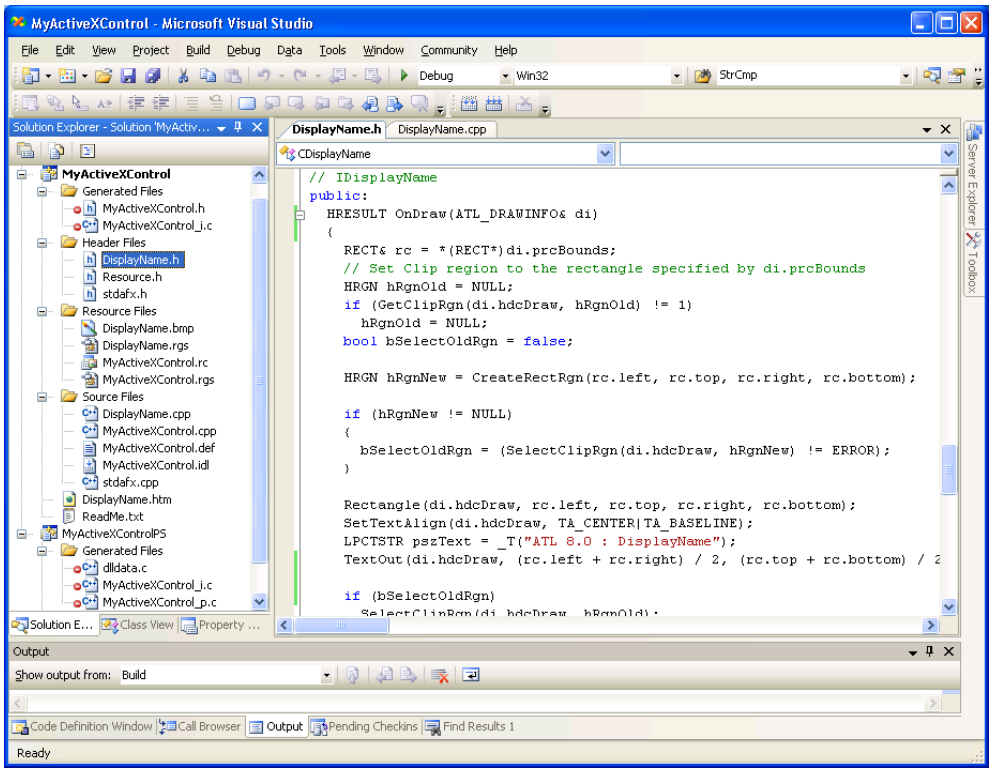


6) The next step is to implement the code created for each new property we added with the Add Property Wizard. Thus, in our example property "Roll", we will need to focus in on the put_Roll and get_Roll functions. In preparation, we will need to add a class member variable for each new property. For the Roll property, add a new class member variable "mRoll" to the class with the same data type as the property (double in this example). Don't forget to set this new member variable mRoll to 0.0 in the "FinalConstruct" function.

7) For the put_Roll function, we will need to capture the incoming value and store it in a class member variable. Once that is complete, we will call a function named "FireViewChange" that will let our display know it needs to be redrawn. When our redraw function (OnDraw) is called, we will then use the value contained in the class member variable to draw the display. In addition, we will set a variable called m_bRequiresSave to TRUE, telling Iads that we wish to save our display. This is a vital step to ensuring your display is saved within IADS when a property is changed. Notice that we only perform this code if the incoming property value is different than the existing value of the property. The get_Roll function is easy to implement. Simply return the value of our class member variable that we have created for this property.

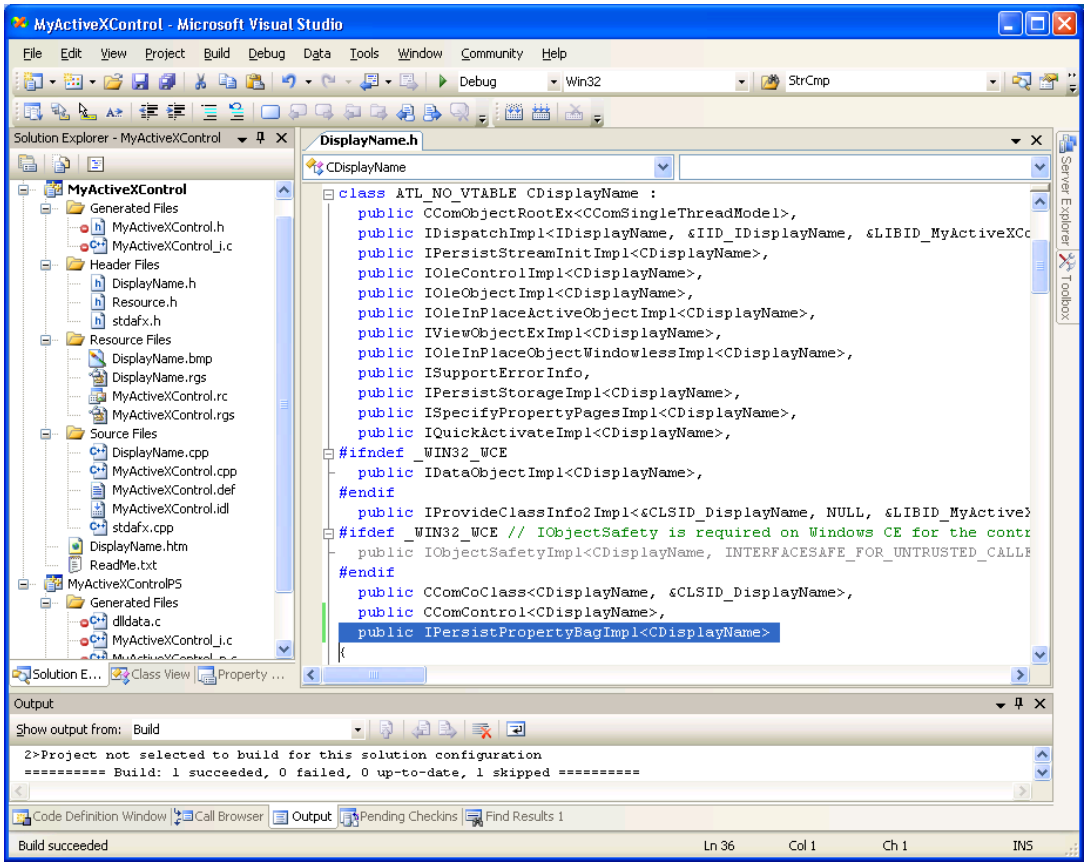


8) Now, when all of your properties are implemented, add your “drawing code” to the “OnDraw” function. The OnDraw function is where you will take all the values of your class member variables and actually draw the display content. See the sample projects for examples in both GDI and OpenGL.

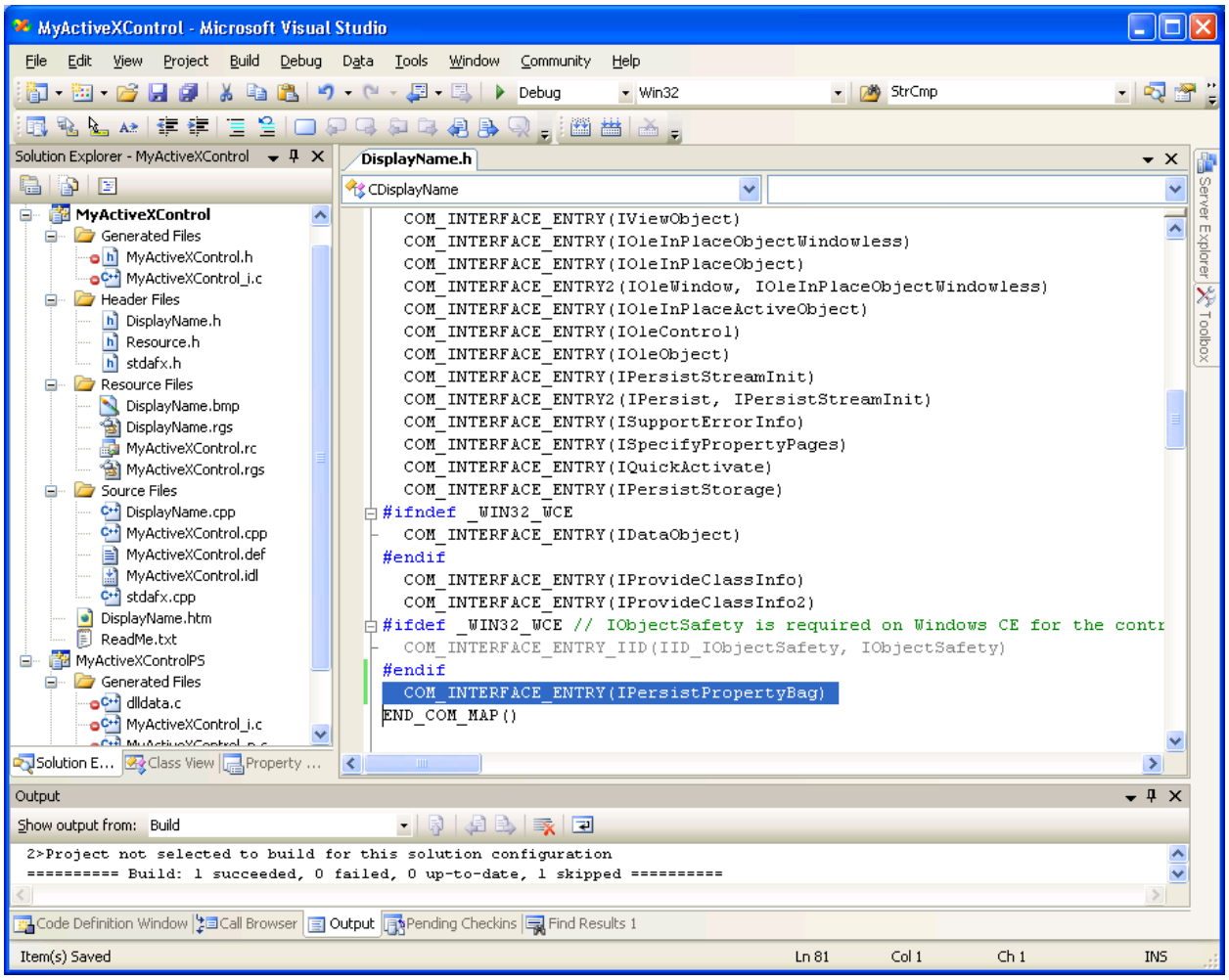


2.2. Ensuring Your Display is Saved Within IADS

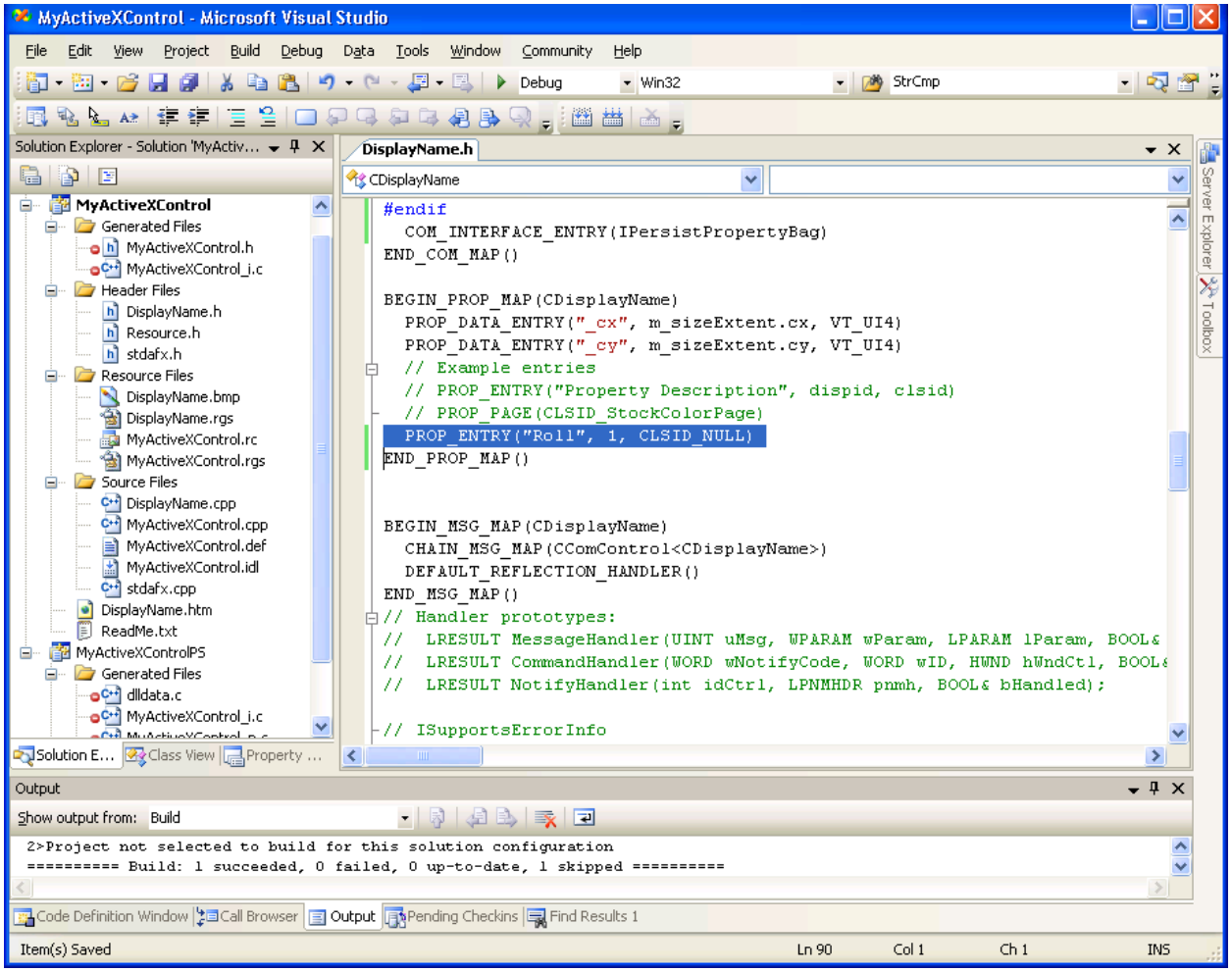
1) Go to your ".h" file (SampleDisplay.h in this example) and insert the code "public IPersistPropertyBagImpl<CYourClassName>" as below. This will allow the control to "save" in IADS properly. Don't forget to add a "comma" to the end of the line above this new line.



2) Likewise, you need to add “COM_INTERFACE_ENTRY(IPersistPropertyBag)” to the “Com Map” in the “BEGIN_COM_MAP” area of the code as below. This is also needed to allow the control to “save” in IADS.



3) For every property to “save”, add it to the “BEGIN_PROP_MAP” area of the code as below. The number corresponds to the property id that’s defined by the wizard in the “.idl” file of the project. Examine your “.idl” file from the “Solution Explorer” tab of the workspace for the correct number. Properties in your “PROP_MAP” will get saved by IADS and reloaded when the display is created in a saved Analysis Window.



4) At this point, you can begin modifying the code in the display to perform your specific needs. For more background on how to build an ActiveX display download the Sample ActiveX Display projects from the Symvionics web site and read the comments within the code:

http://iads.symvionics.com/MainPages/DownloadsPage.htm

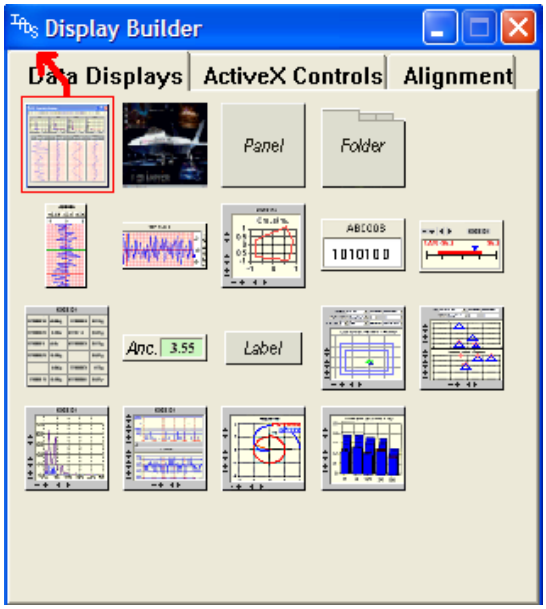
If you have any further questions, you can search the Iads Google Group or post a question:

http://groups.google.com/group/iads

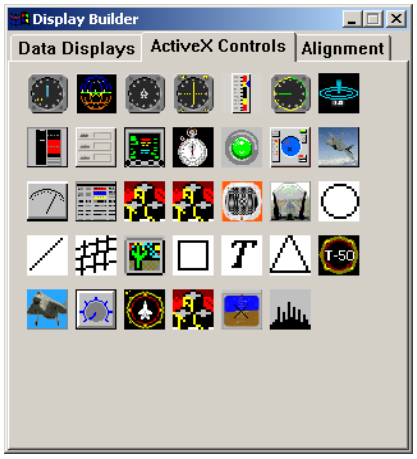
3. Adding Your New Control to IADS

1) Press the “Display Builder” button on the IADS “Dashboard” in the lower right hand corner of the screen. The “Display Builder” dialog will appear with icons of components that you can use to build your displays (including your new ActiveX control).

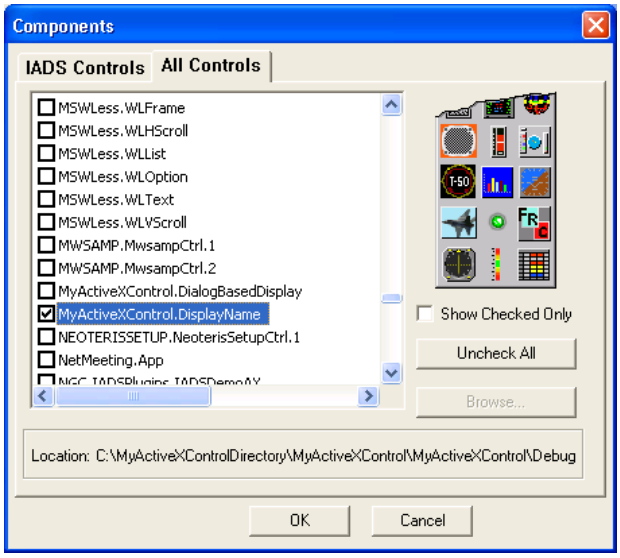
2) Create an empty Analysis Window by dragging the upper left icon on to your desktop and dropping it. You can optionally name your window here.



3) Now let's add your new control to the "Display Builder". Click on the second tab in the display builder named "ActiveX Controls". This is where all ActiveX displays will reside, ready to be dropped upon your newly created Analysis Window. Notice that there are only a select few ActiveX control icons on this tab of the display builder. If the display builder were to show all of the controls available on your system, the icons would fill several pages of this size. In order to add your new control, you must "right click" on tab (somewhere where there are no icons). This will activate yet another dialog containing both the "IADS" supplied ActiveX controls as well an entire list of all the ActiveX controls on your system (including your newly created one!). Click on the "All Controls" tab of this new dialog and find your new control. The name will be the "ProjectName.ObjectName" as discussed earlier in this tutorial. In the example code, my control is named "MyActiveXControl.DisplayName.1" (.1 is the version). Click "Ok" to add your display to the display builder. This only needs to be done once for each new control that you wish to debug/add in IADS.

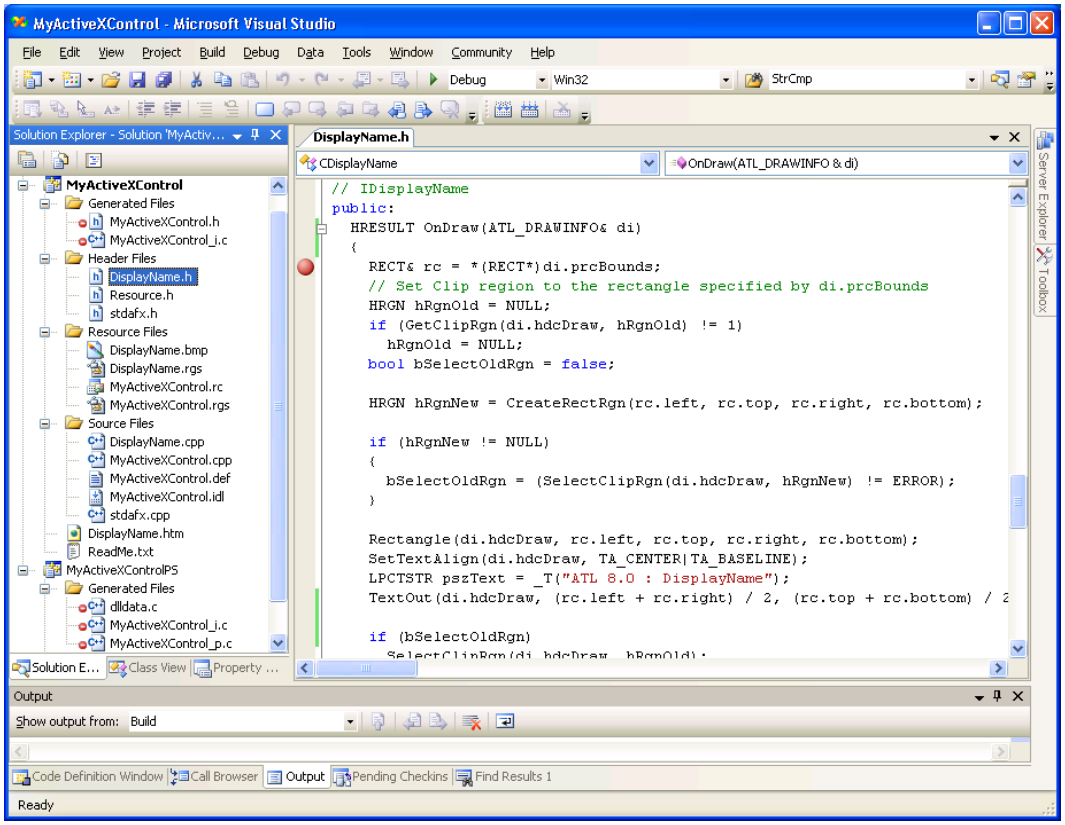


Note: If you can't locate you display in the "All Controls" list, try checking your ".rgs" file in your VS2005 project in the "workspace" fileview. It contains an entry named VersionIndependentProgID. This is where VS2005 stores your "ProgId" (Program ID) for the project.

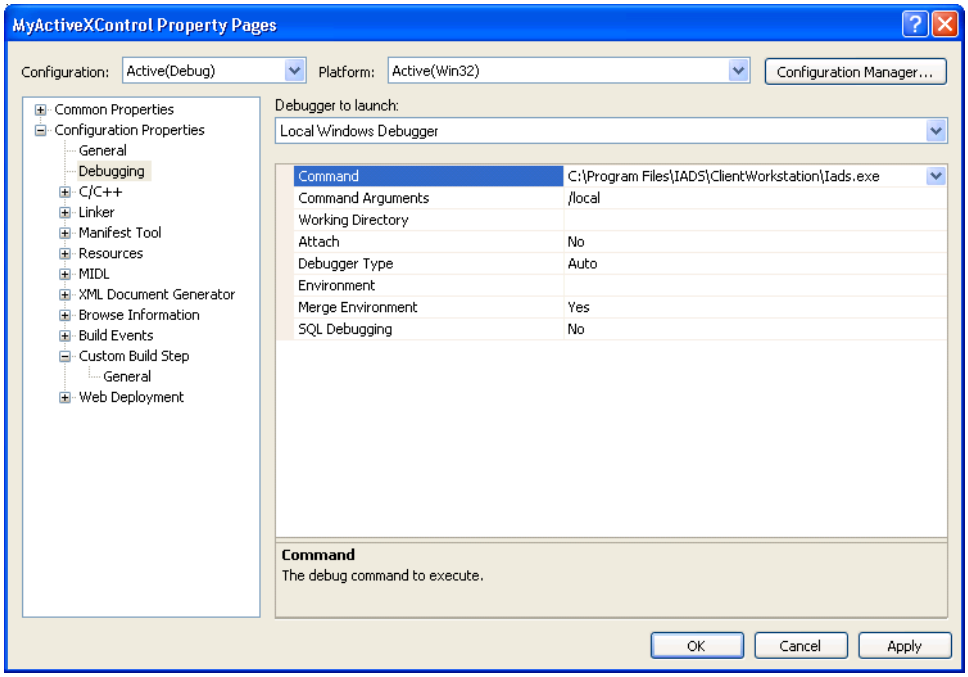


4. Debugging Your New Control in IADS

1) Place a break point in your “OnDraw” method for testing.



2) In Visual Studio, select the “Project->Properties” drop down menu. Under the “Configuration Properties->Debugging” tier, pick “Iads.exe” as your “Command”. The Iads.exe is located in your “C:\Program Files\Iads” directory. Add “/local” to your “Command Arguments”. Build your project and click on the “Go” command. Iads will start.

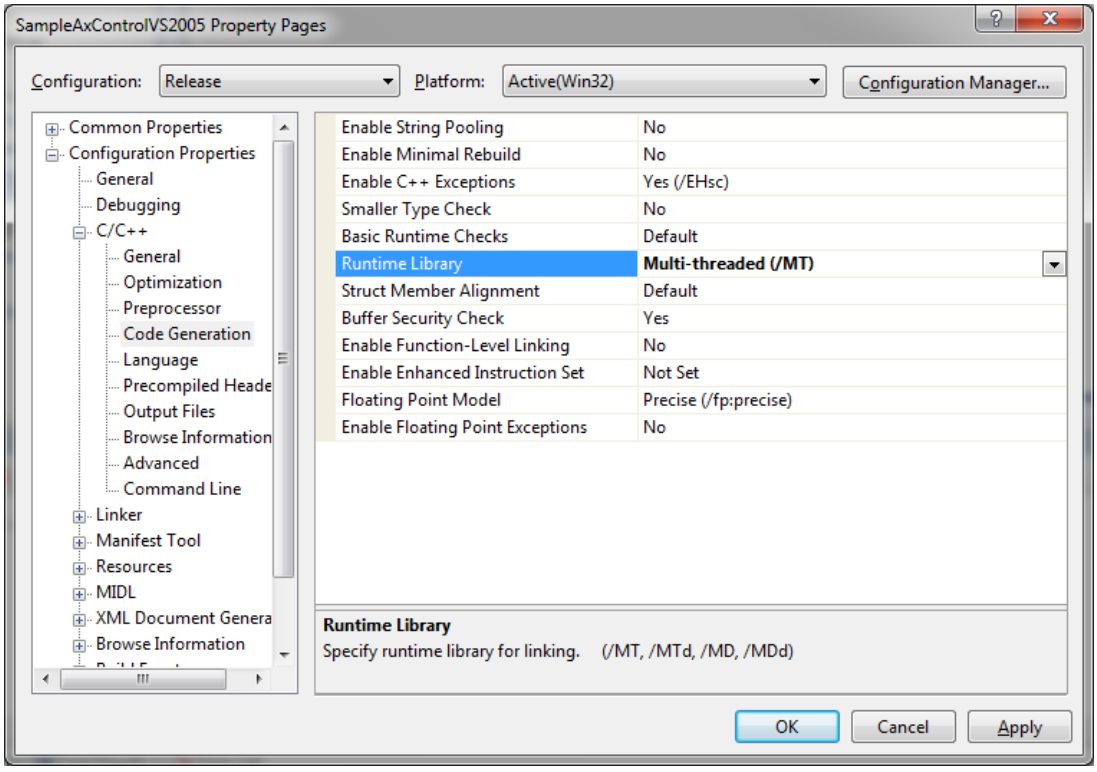


3) Drag-n-Drop your display to the new Analysis Window as explained previously. Your break point should now hit in the debugger. You can now step through your rendering code if necessary.

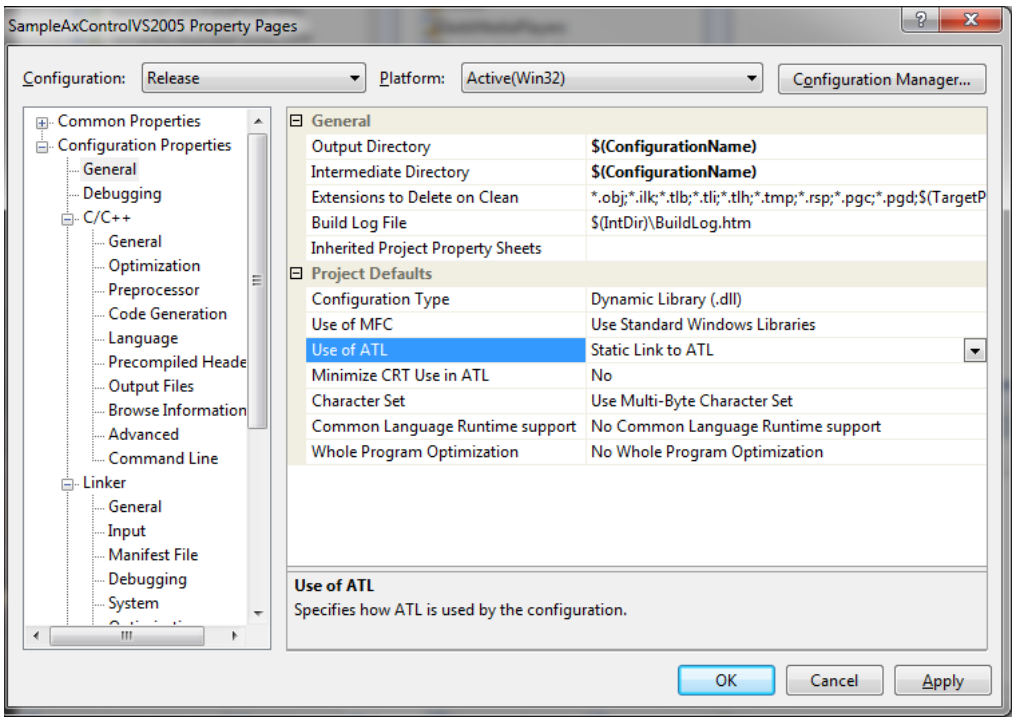
5. Deploying your New Control

When it comes time to deploy your new control to users on other PCs, you need to consider a couple of issues. One issue is that your control may require some auxiliary dlls that are not available on the other systems. If that occurs and the dlls are missing, the control may not operate. To help minimize this possibility, you must always build your new control dll in "Release" mode. You should never distribute a control dll that has been compiled under the "Debug" mode. The debug mode uses libraries that will most certainly be missing on any machine without Visual Studio installed. Beyond that, it's always best to 'statically link' all the runtime libraries. Also, since we've used ATL to build this display, we will need to statically link the ATL library as well.

1) In Visual Studio, select the "Project->Properties" drop down menu. Make sure that the "Configuration" drop down is set to "Release". Under the "Configuration Properties->C/C++->Code Generation" tier, set the "Runtime Library" to "Multi-threaded (/MT)".



2) Under the “Configuration Properties->General” tier, set the “Use of ATL” to “Static Link to ATL”.



3) Once you’ve made these changes to your project, you should rebuild your ‘solution’. Make sure once again that your current configuration is set to “Release” and then select the “Build->Rebuild Solution” drop down menu option. After this step is complete, your control dll should be in your project “Release” folder. It should now be ready to deploy on another system.

   The control dll will need to be copied to the other PC and ‘registered’. In order to register the dll, you’ll have to run the ‘regsvr32.exe’ program. One easy way to accomplish this is to double click on the dll in Windows Explorer. When asked what program to execute on the dll, navigate to the Windows\System32 directory and choose the regsvr32.exe file. This procedure may be different if the operating system is a 64 version. Please consult the online documentation for specifics.

   If the dll fails to register at this point, we’ve most likely failed to statically link the needed dlls. We can investigate which dlls are missing by using the “Dependency Walker” tool. The Dependency Walker program is located within the Microsoft Visual Studio\Common\Tools directory and is named “Depends.exe”. Copy Depends.exe from your development PC to the target PC and run the program. From the File drop down menu select “Open” and choose your control dll. Examine the module list in the bottom window pane. Any missing dependent dlls should show up with a question mark. Search for those dll names on the net and find out their purpose. It might help you narrow down what solution setting you’ve missed. It is also possible that the missing dll is a private library that you are using, in which case you’ll need to either static link or copy that dll to the target machine as well.