Creating an IADS Custom ActiveX Control using Visual C++ 6.0

January 2010

SYMVIONICS Document SSD-IADS-051

SYMVIONICS, Inc.
Telemetry Systems

Table of Contents

1. Introduction

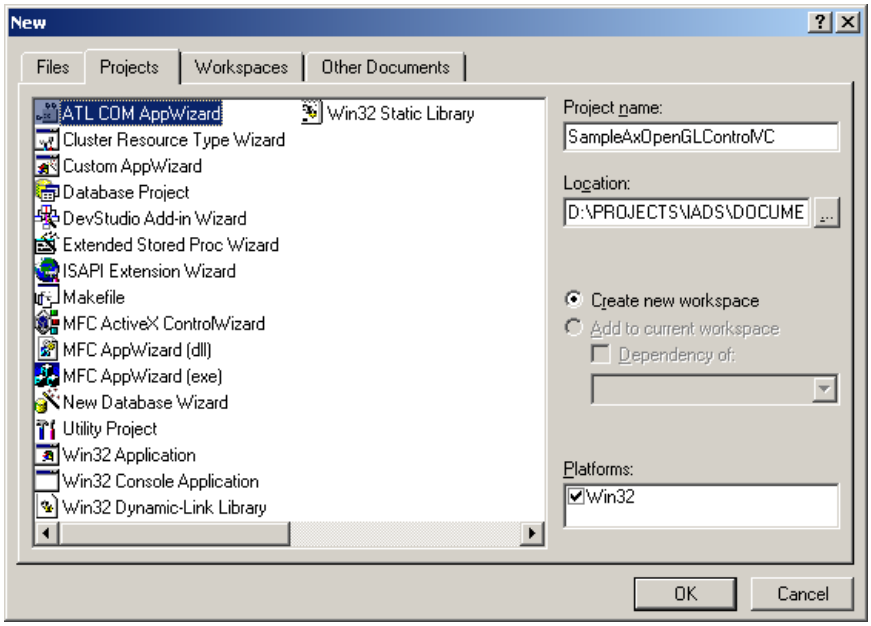
This document assumes you are using Microsoft Visual C++ 6.0. The tutorial has not yet been attempted on a newer version, although it may still apply. This instruction guide will cover: creating a new display using the ATL COM Wizard, adding the new control to IADS, debugging the new control in IADS, and ensuring the new display is saved within IADS.

2. Creating Your Display using the ATL COM Wizard

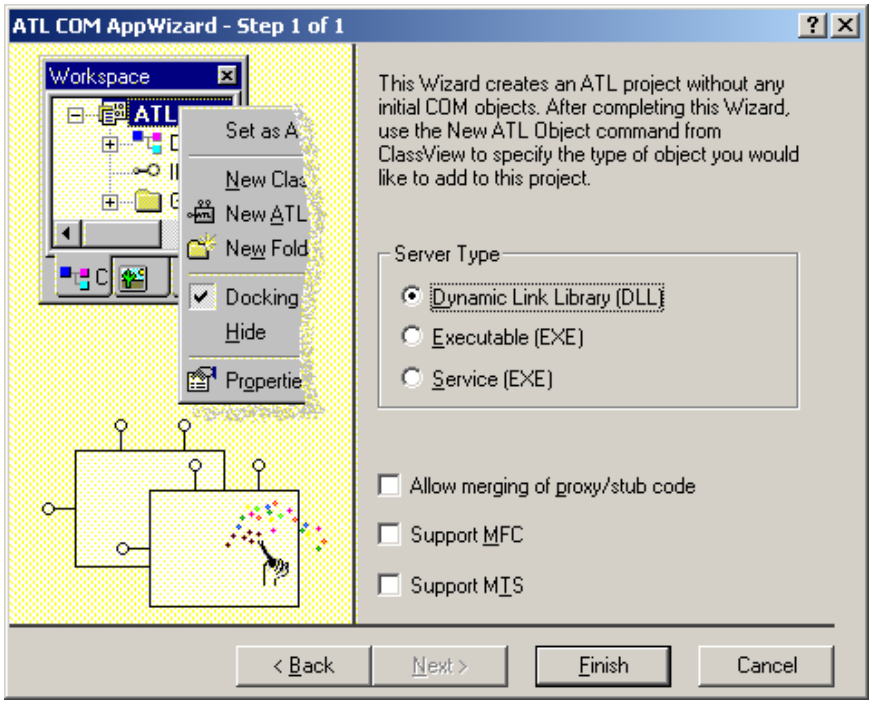
Select "File -> New -> Projects Tab -> ATL COM AppWizard" and pick your project location and name. The project name you choose will become part of the display identifier name (aka ProgID, see inset). When it comes time to use your control in IADS, users will insert your new control into the "Display 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).

Microsoft refers to your control's name as its "ProgID" (aka Program ID). This is the string equivalent of your GUID (Global Unique Identifier) for the control. 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).

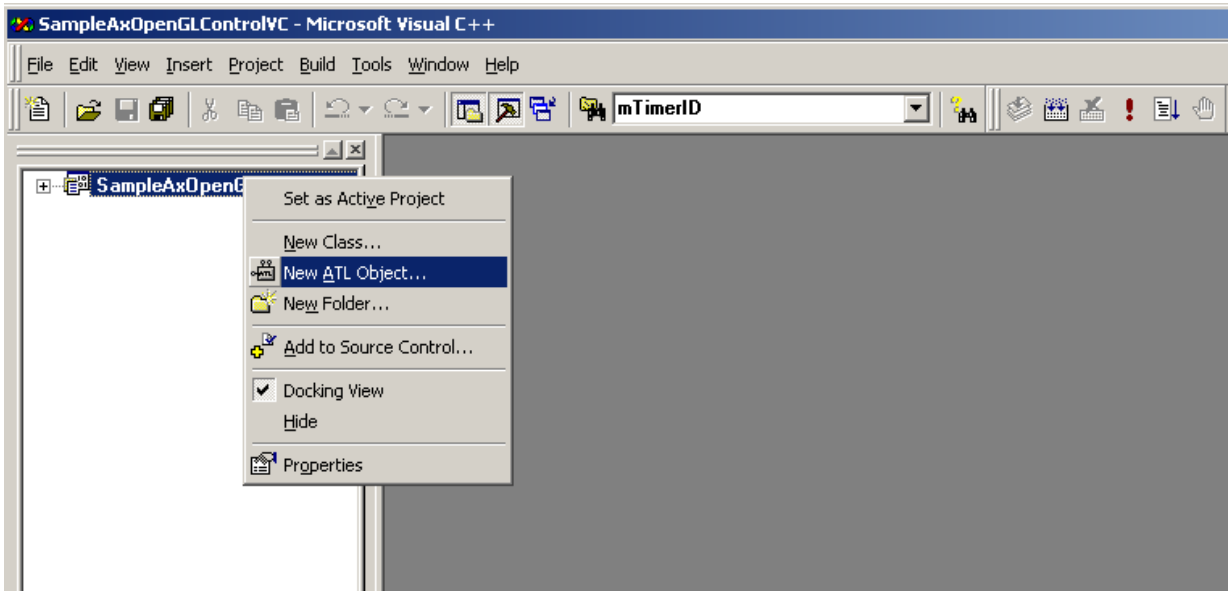
1) Choose wisely and click OK when you are ready to continue.



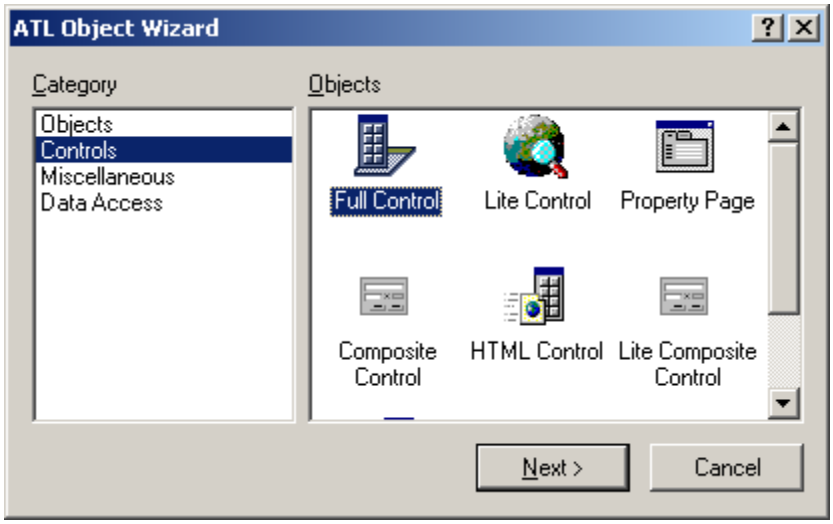
2) After clicking “Ok”, the “ATL COM AppWizard” dialog will appear as below. In step 1, choose “Server Type” DLL and click Finished. Most every ActiveX display that runs in IADS is of type “DLL”. This allows for maximum speed while drawing and performing calculations.



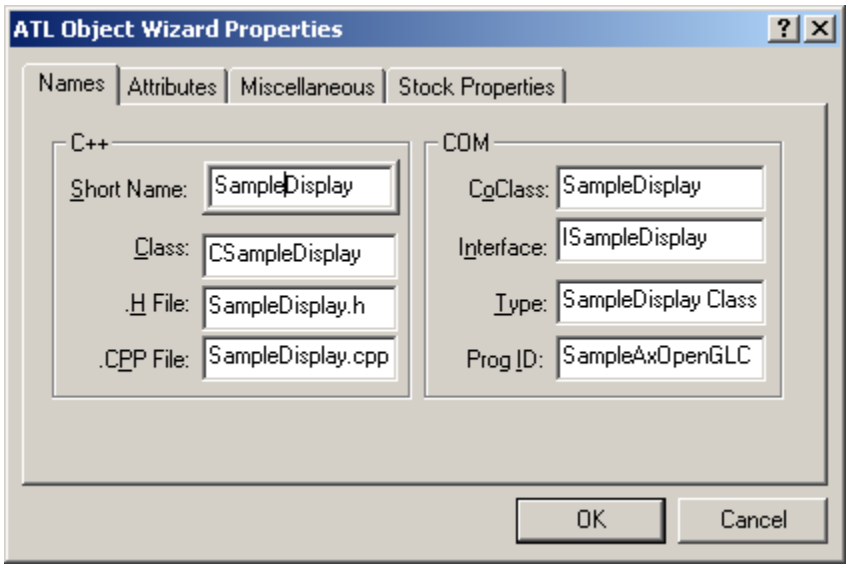
3) Next, go to the “ClassView” tab in DevStudio’s workspace view and right-click on the project name. Choose “New ATL Object”.



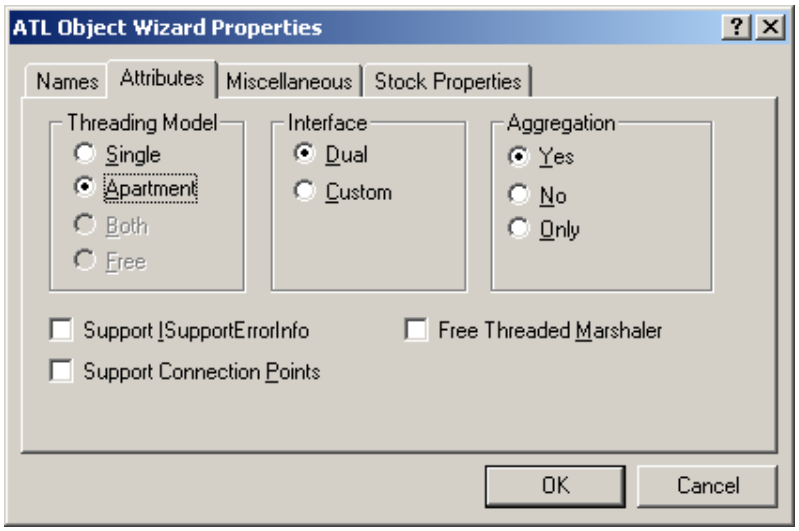
We want the wizard to "auto-create" a full ActiveX control in your project. When you are finished with the steps of this wizard, it will create amazing amounts of complicated ActiveX template code; saving you time and sanity. To master this process, eventually, you will have to understand most of the code the wizard creates. We'll take this on at a later date. For now, let's just focus on the basics and create a new control. This will go quicker than you think.



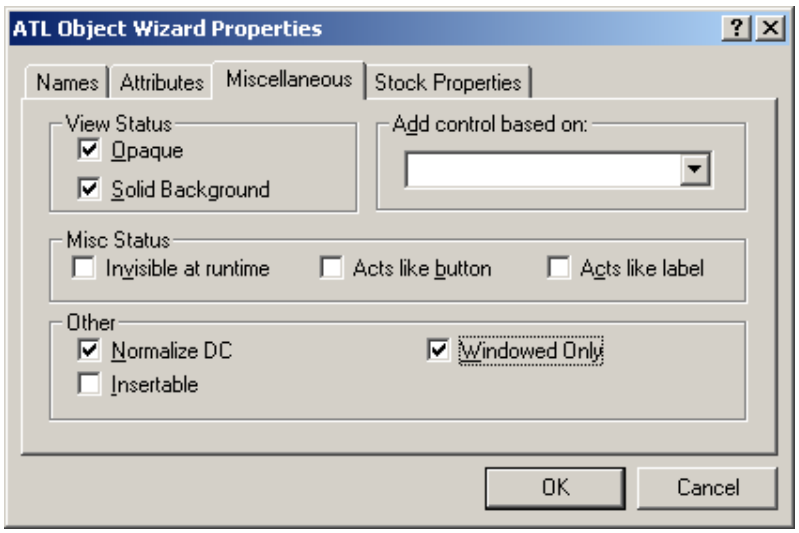
4) Select "Controls" in left area, and "Full Control" on the right…and then click Next. On the first tab, enter the name of your display (control) in the "Short Name" field. The wizard will fill out the rest of the tab automatically. For this example, I used "SampleDisplay" as the short name. The name entered will be the "class name" of your display, so choose a descriptive name like "AttitudeIndicator" or "FluidSystem13" that describes your display well (aka Species). Again, this name combined with your "project name" will be presented to the users of your display as its "ProgId" as explained on page 1. See the "ProgID" field in your dialog for your final control name.



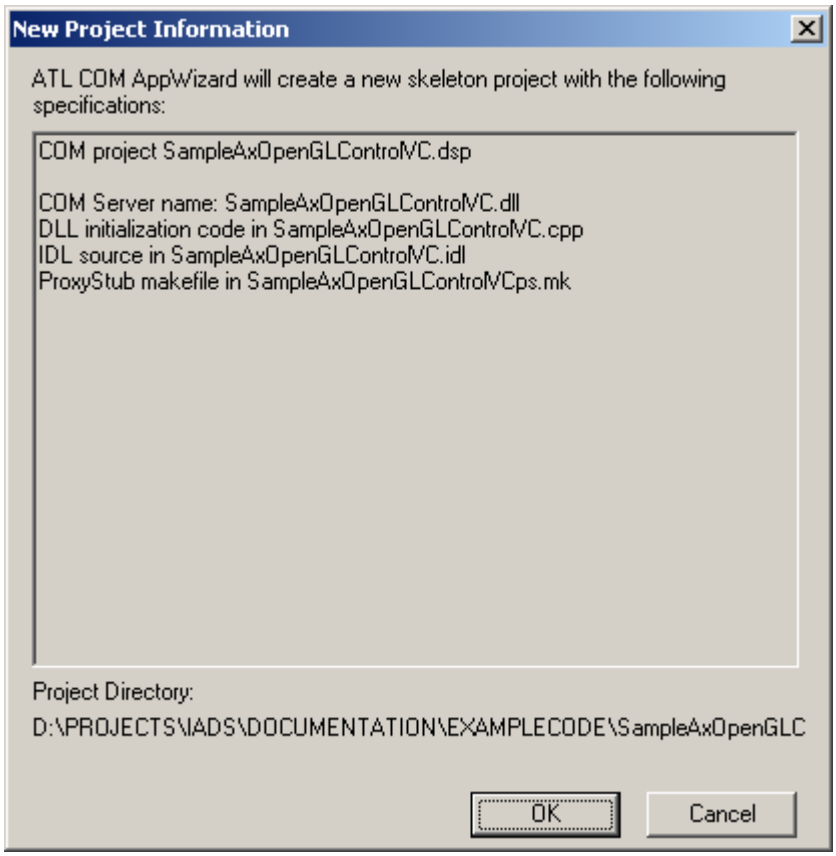
5) On the next tab (“Attributes”), leave everything as default (Apartment, Dual, Yes, and no other). These options are basically COM “speak”. If want understand this fully, you’ll have to consult the Microsoft documentation. The most important here is “interface”; in order to create a real compliant ActiveX “display”, you must choose “Dual” interface. This will allow IADS (and other programs) to interface to your control using the “IDispatch” interface, which allows loosely coupled, “on the fly” communication (see Microsoft documentation). This also happens to be the primary (simplistic) way that IADS gets data to your control. More on this subject later.



6) On the next tab (“Miscellaneous”), 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). At this point you can click OK (leaving “Stock Properties” for later).

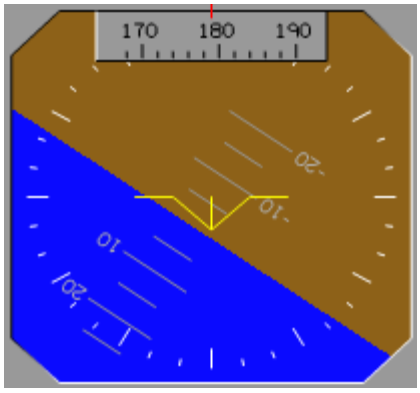


7) After clicking “Ok”, you’ll get a window like this from the Wizard showing what files have been auto-created. Examine your “Workspace” view… It should now contain the new “Full Control” object by name. Click “Ok”. You’re half way home now.

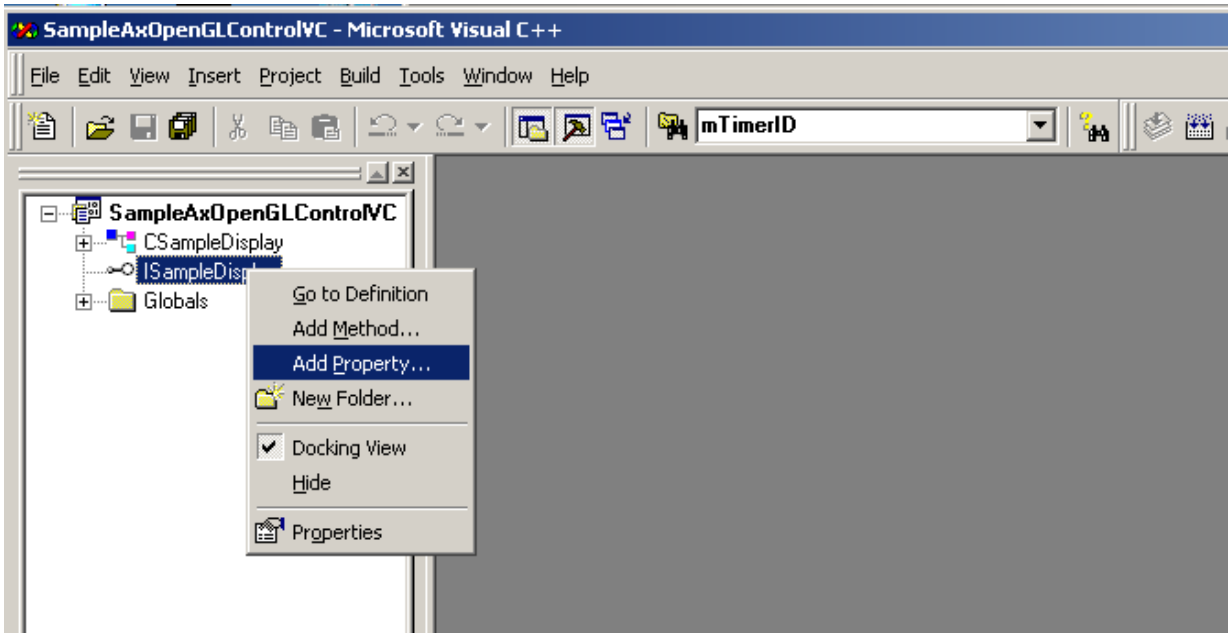


2.1 Adding Properties to the Display

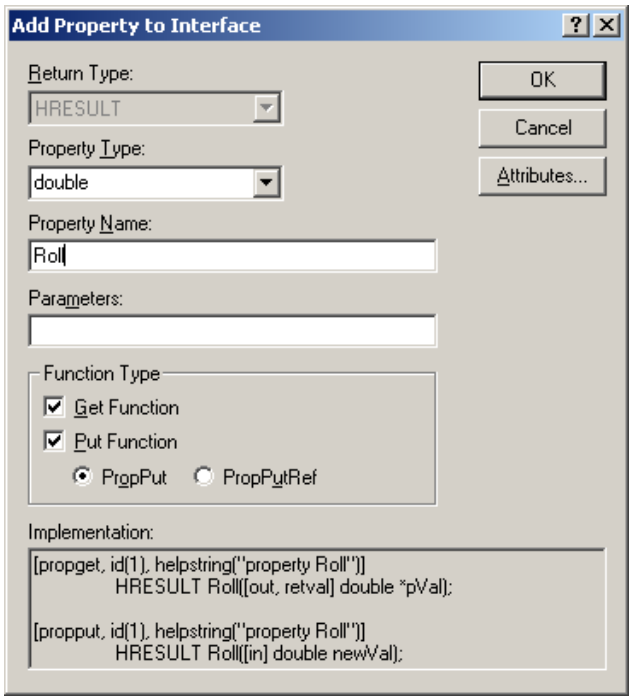
Now it's time to add "Properties". Think of properties as "data injection ports" or "interface plugs". They are really just any attributes of your display (such as text color, needle angle, or scale factor) that you want the user to be able to change or animate. To give a concrete example, the included 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 the 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.



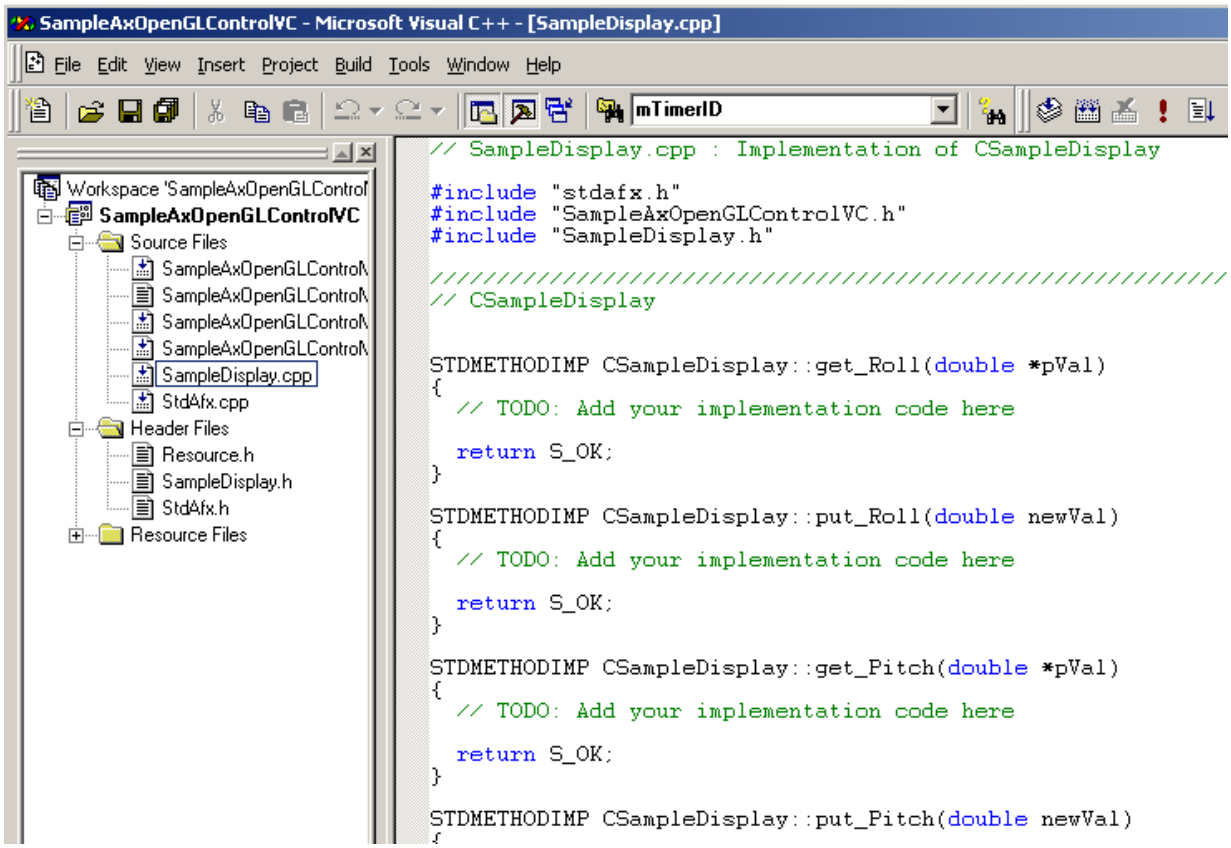
1) Now, make sure that you are in the "ClassView" tab of the VisualC++ workspace. To add a "Property" to your new control, right click on the "IXXXXX" where "XXXXX" is the name of your newly created control (look for the little "magnifying glass" icon). Select "Add Property" from the popup menu.



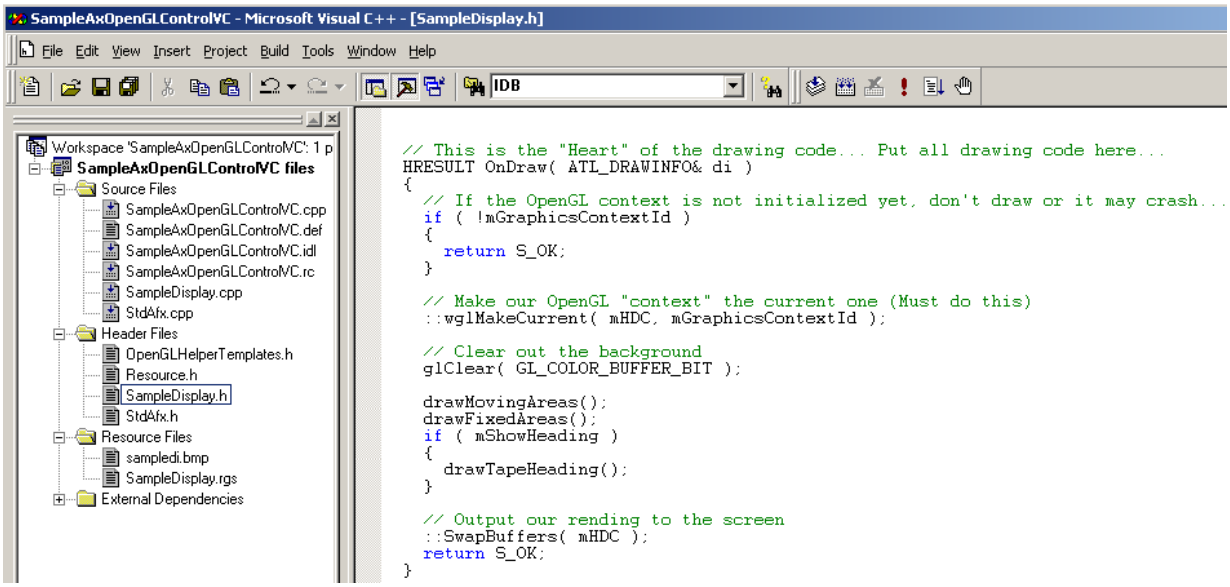
2) Set property type (double in this example) and type in the name of the property (Roll in this example) and click "Ok". This will auto-create code to implement a property named "Roll" within your new project. Repeat this process for every property.



3) Going back to your “FileView”, you’ll see the new code inserted.

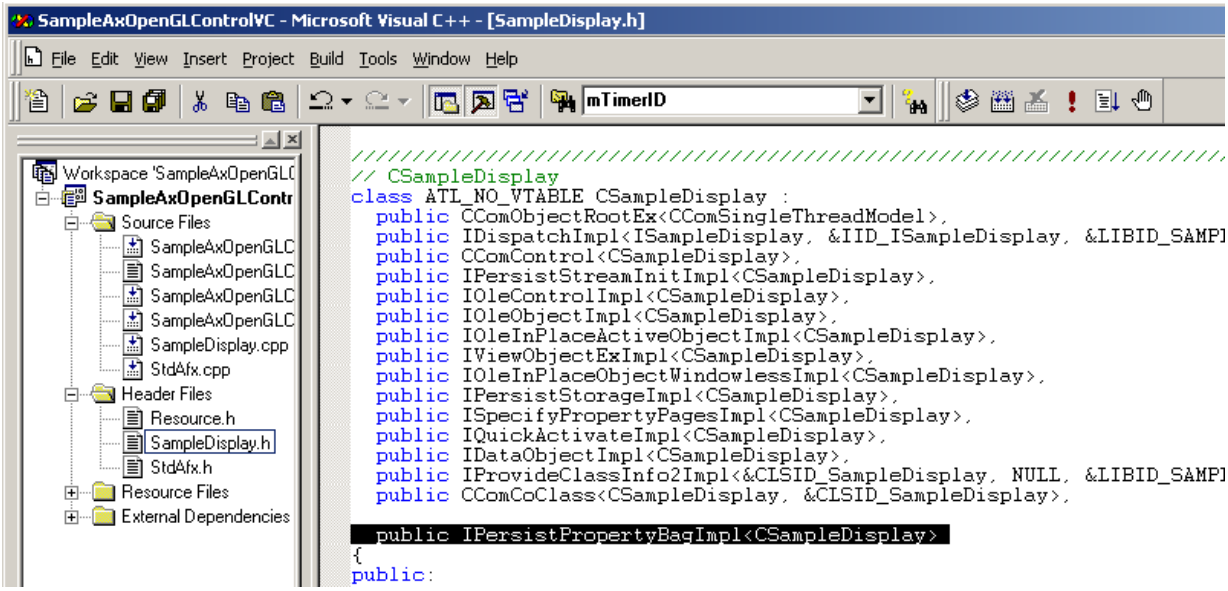


4) Add your “Rendering code” to the “OnDraw” method. See sample project for details.

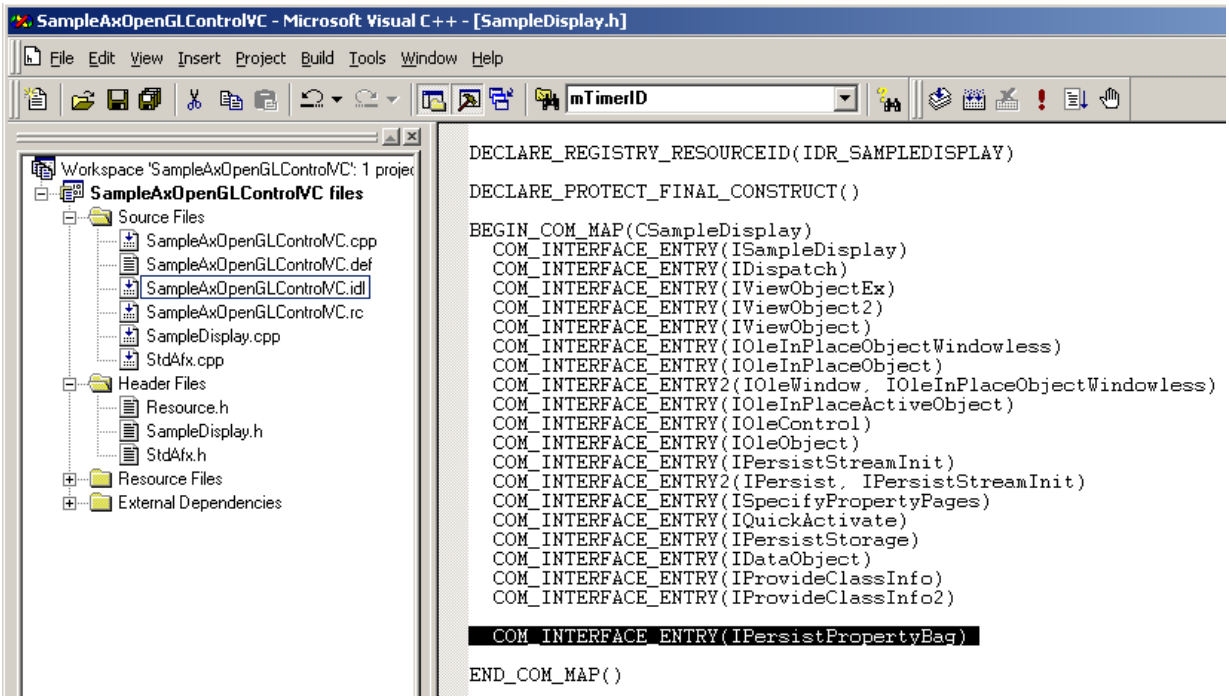


2.2 Ensuring Your Display is Saved Within IADS

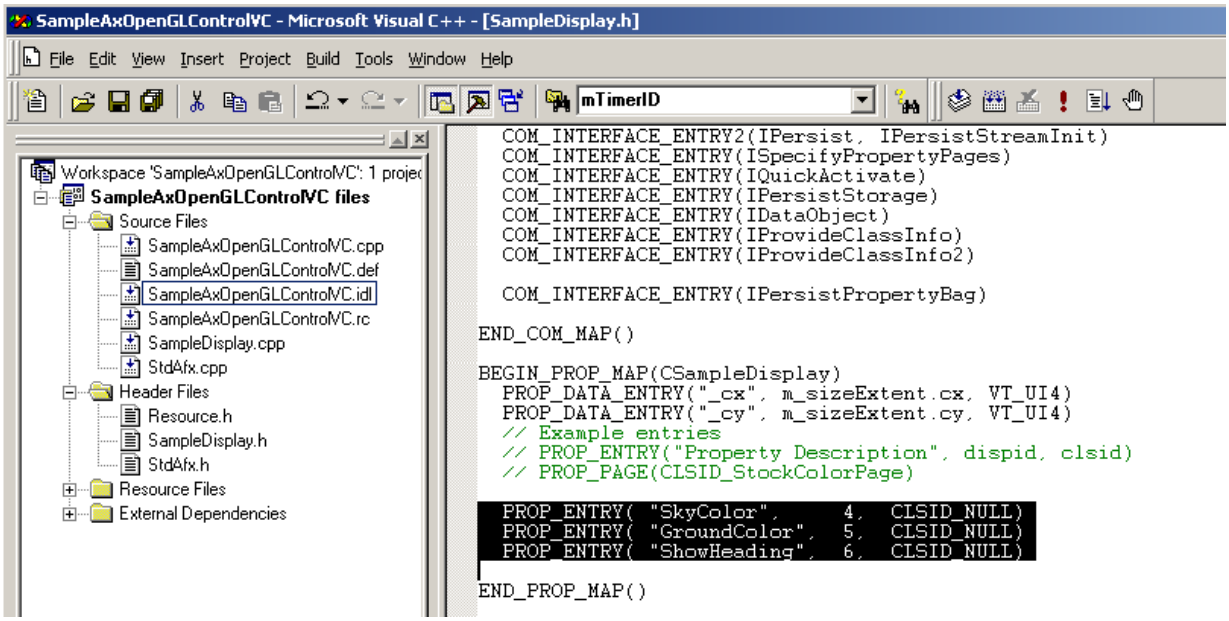
1) Go to your ".h" file (SampleDisplay.h in this example) and insert the code "public IPersistPropertyBagImpl<CYourDisplayNameHere>" as below. This will allow the control to "save" in IADS properly. Don't forget to add a "comma" to the end of the line above this new line.



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 "FileView" 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:

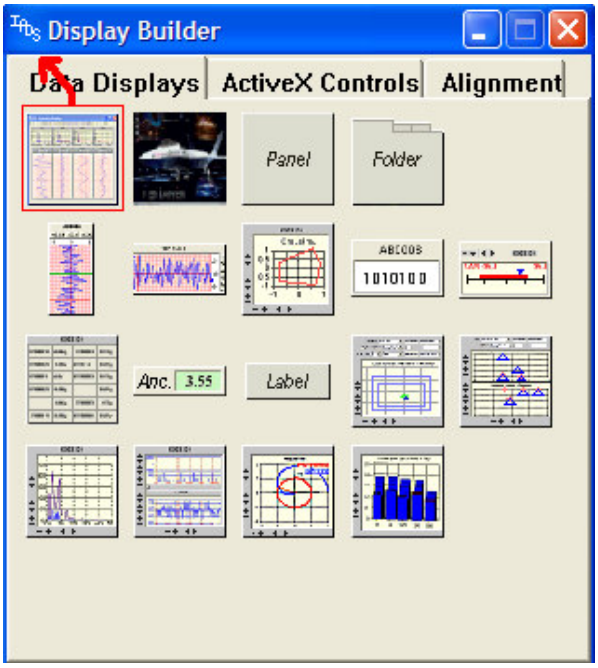
http://groups.google.com/group/iads

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.

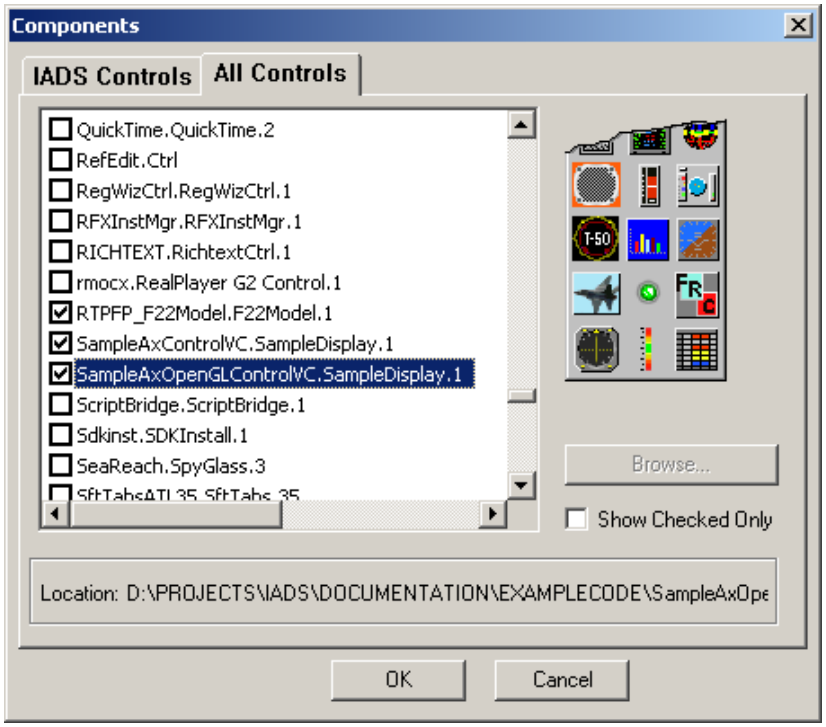
The “Display Builder” dialog will appear with icons of components that you can use to build your displays (including your new ActiveX control).

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



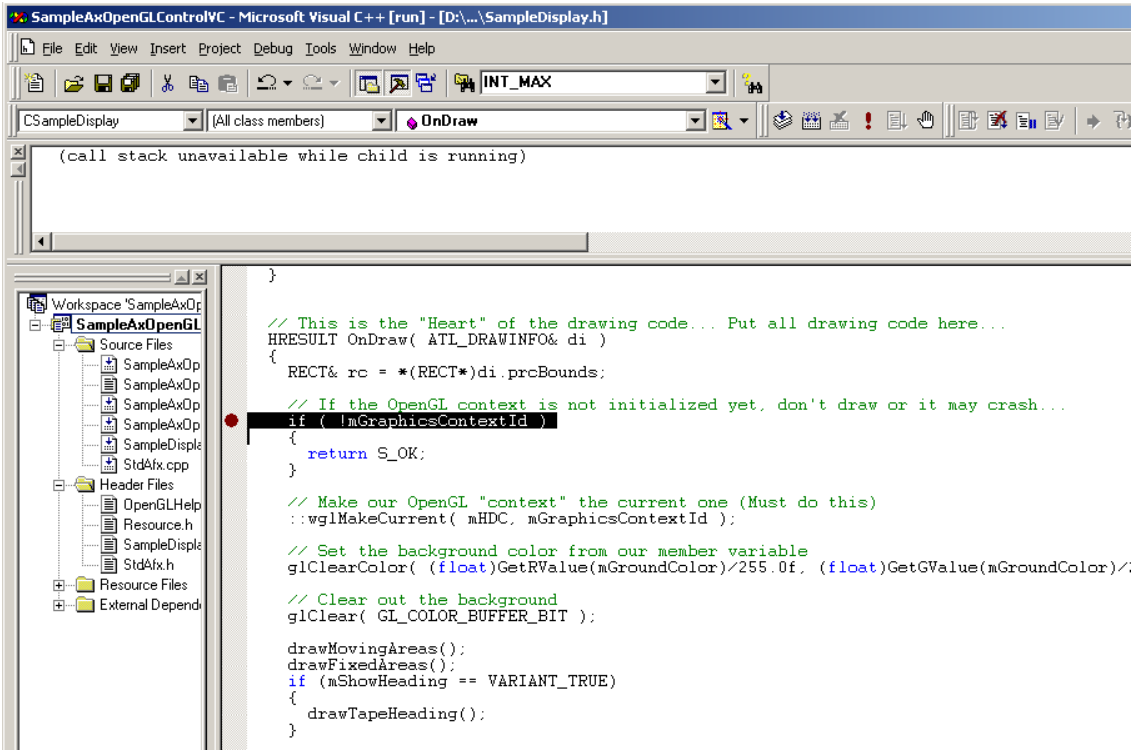
3) Now let’s add your new control to the “Display Builder”. Click on the second tab in the display builder named “ActiveX Controls”. This is where all ActiveX displays will reside, ready to be dropped upon your newly created Analysis Window. Notice that there are only a select few ActiveX control icons 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 “VisualC++ProjectName.ObjectName” as discussed earlier in this tutorial. In the example code, my control is named “SampleAxControlVC.SampleDisplay.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 VisualC++ project in the "workspace" fileview. It contains an entry named VersionIndependentProgID. This is where VisualC++ stores your "ProgId" (Program ID) for the project.

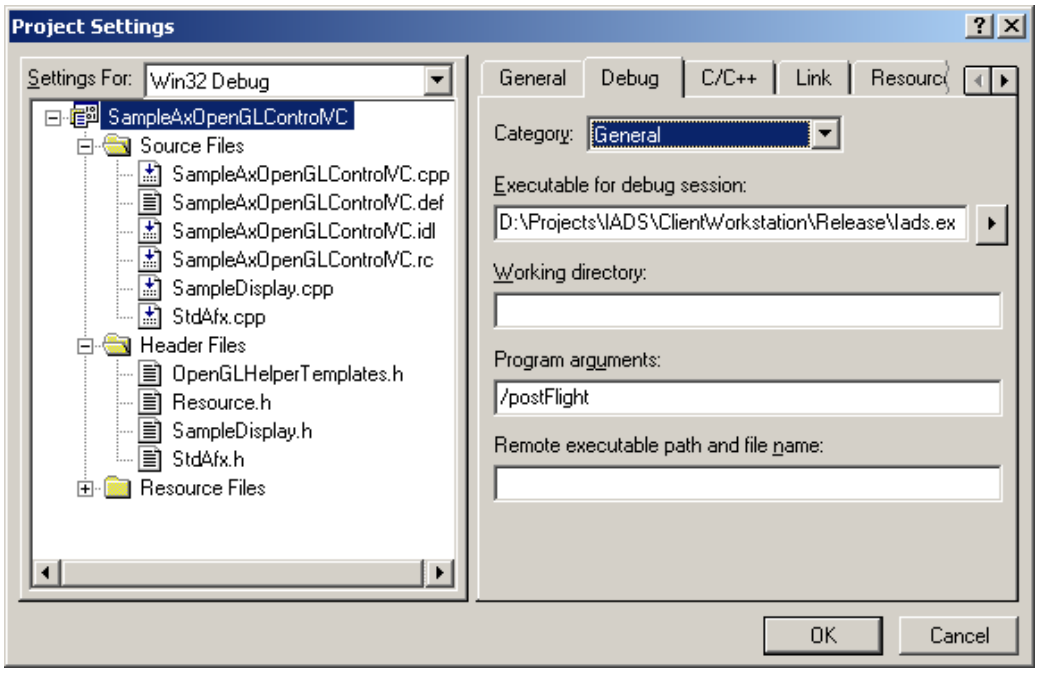


4. Debugging Your New Control in IADS

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



2) Go to “Project Settings” and pick “Iads.exe” as your “Executable for debug session”. It’s in your “C:\Program Files\Iads” directory. Add “/postflight” to your “Program arguments”. Build your project and click on the “Go” (Run from DevStudio F5) command. Iads will start.



3) Add your 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.