Creating an IADS Custom ActiveX Control using C# VS2005

March 2011

SYMVIONICS Document SSD-IADS-145

SYMVIONICS, Inc.
Telemetry Systems

Created March 23, 2011

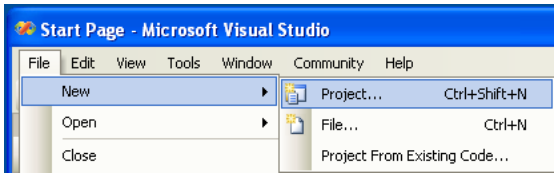
Table of Contents

1. Introduction

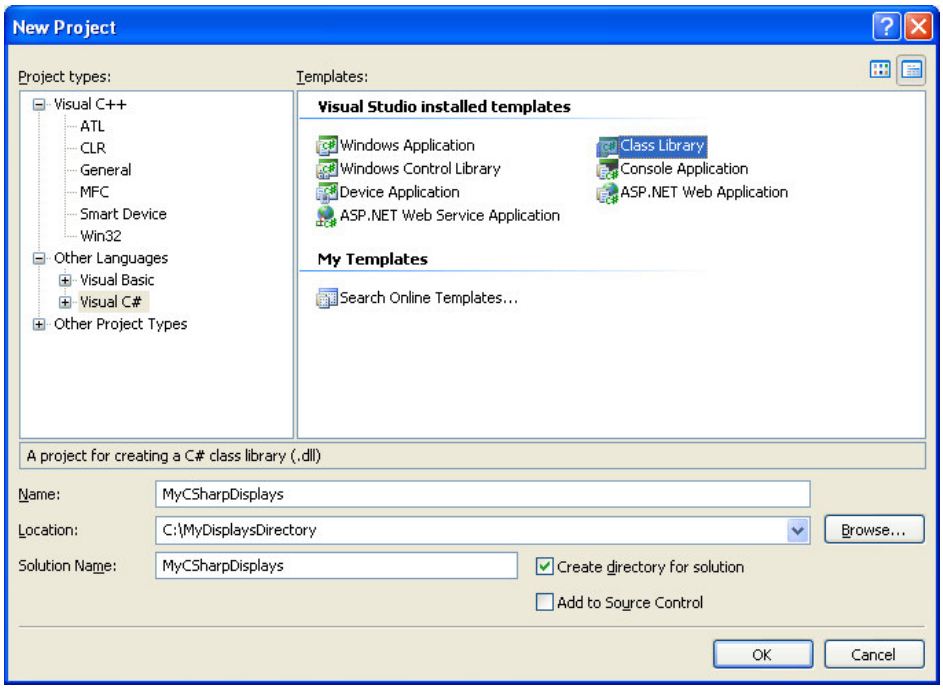
This document assumes you are using Microsoft Visual Studio 2005. The tutorial has not yet been attempted on a newer version, although it may still apply. This instruction guide will cover: creating a new display using C#, adding the new display to IADS, debugging the new display in IADS, and ensuring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

2. Creating Your Display using the Project Wizard

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



2) In the New Project dialog that appears, choose the "Other Languages->Visual C#" tier and click the Class Library" 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.



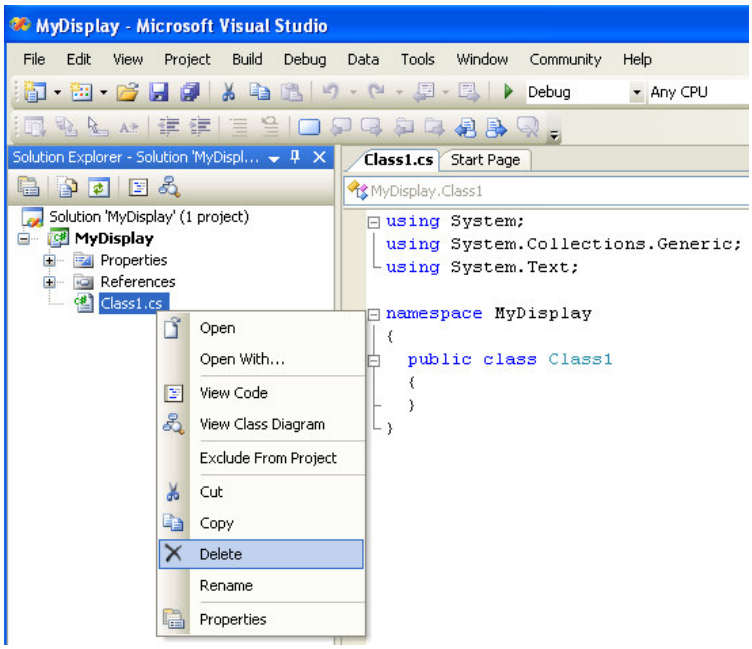
3) The project name you choose will become part of the display identifier name (aka ProgID, see note below). When it comes time to use your control in IADS, users will insert your new control into the "Display Builder" toolbox based solely upon its name (more on this later). Plan on creating many displays 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).

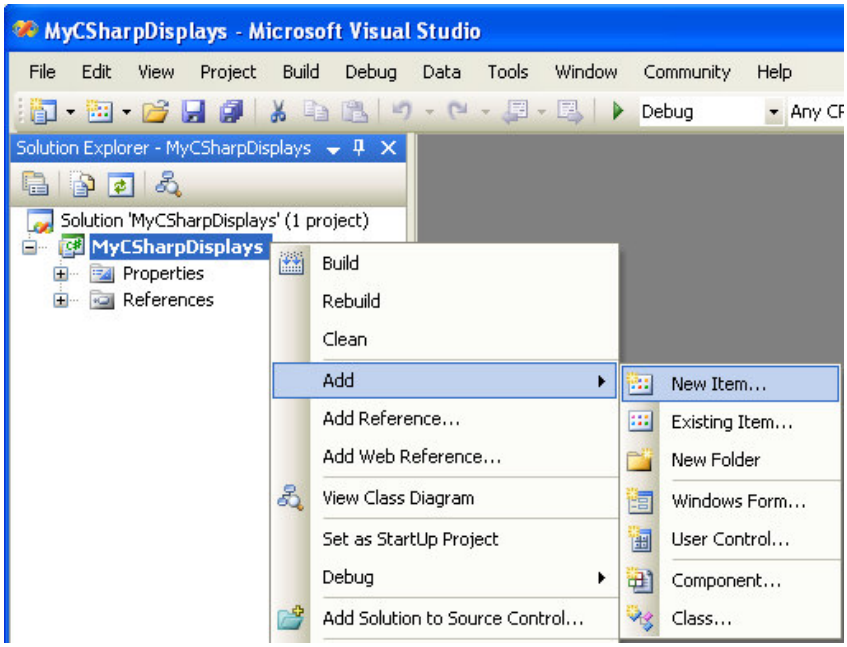
Note: Microsoft refers to your displays’ name as its “ProgID” (aka Program ID). This is the string equivalent of your GUID (Global Unique Identifier) for the display. These Ids are placed in the Microsoft registry, allowing your object to be created without any knowledge of the location of your “Dll” on the file system. Of course, this assumes that it is registered using the “RegAsm” program (consult the Microsoft documentation).

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

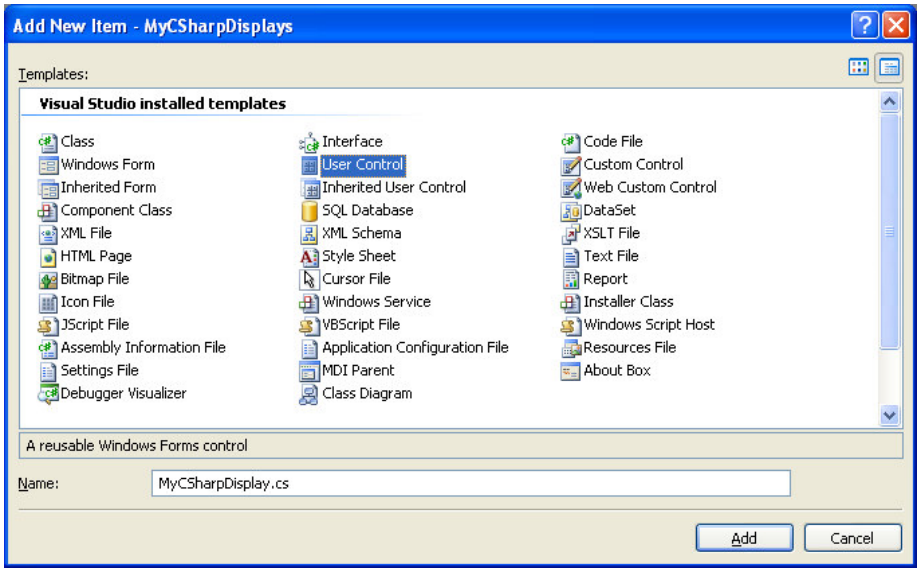
4) After pressing OK, the project will be ready for editing. Before we begin, delete the Class1.cs file. We will not need the file.



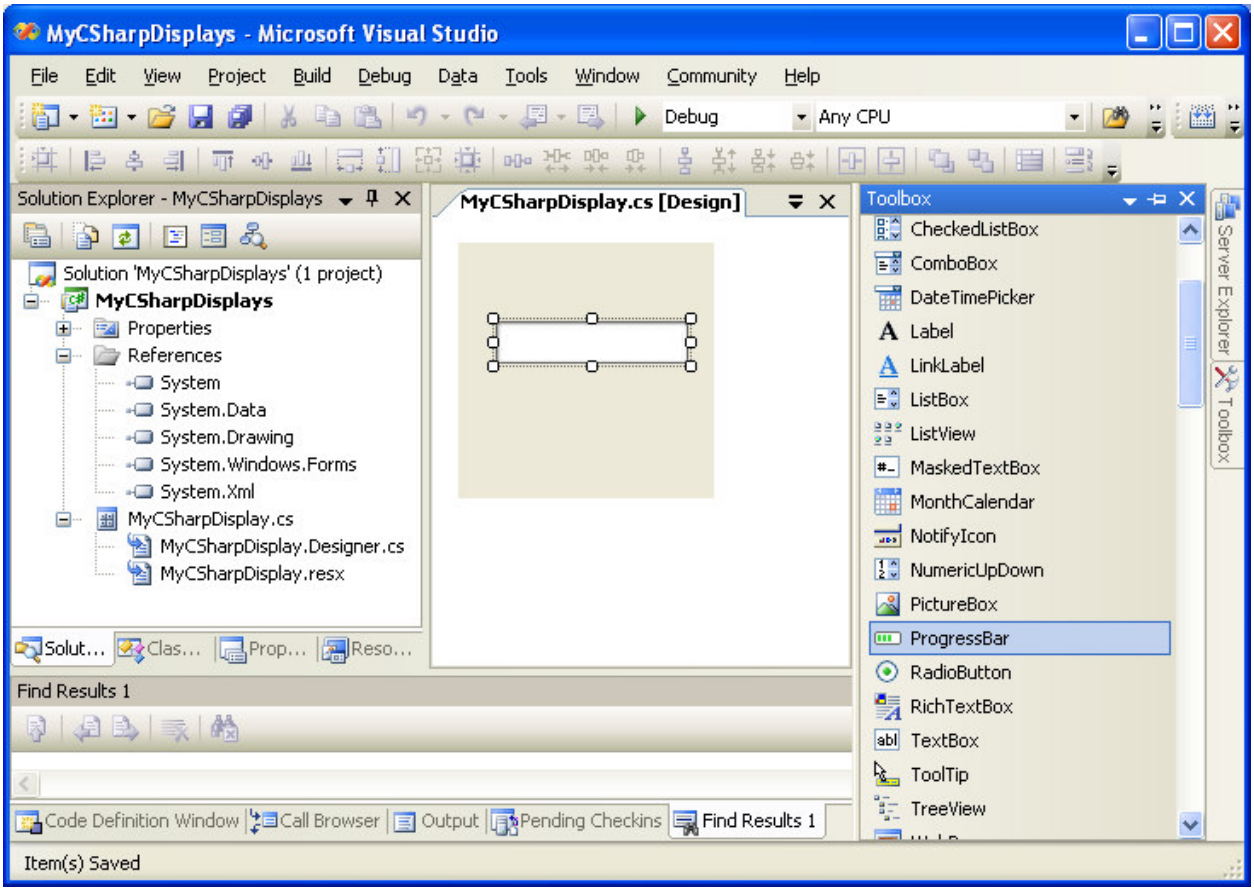
5) Right Click on your project in the Solution Explorer and choose “Add->New Item”



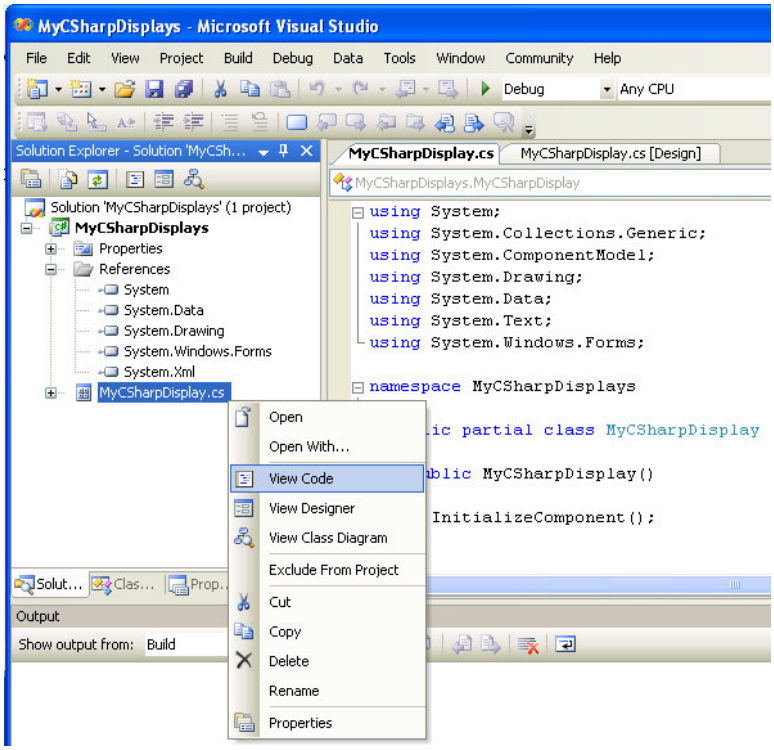
6) In the “Add New Item” dialog, choose “User Control”. Enter the name of your display in the field and click the “Add” button.



7) Visual Studio now displays the “Design” view of the display and shows a blank form. At this point, you can use the Visual Studio “Toolbox” to add a visual object. In this simple example, let’s drop a ProgressBar into the form. It is located under the Common Controls tier in the Toolbox.

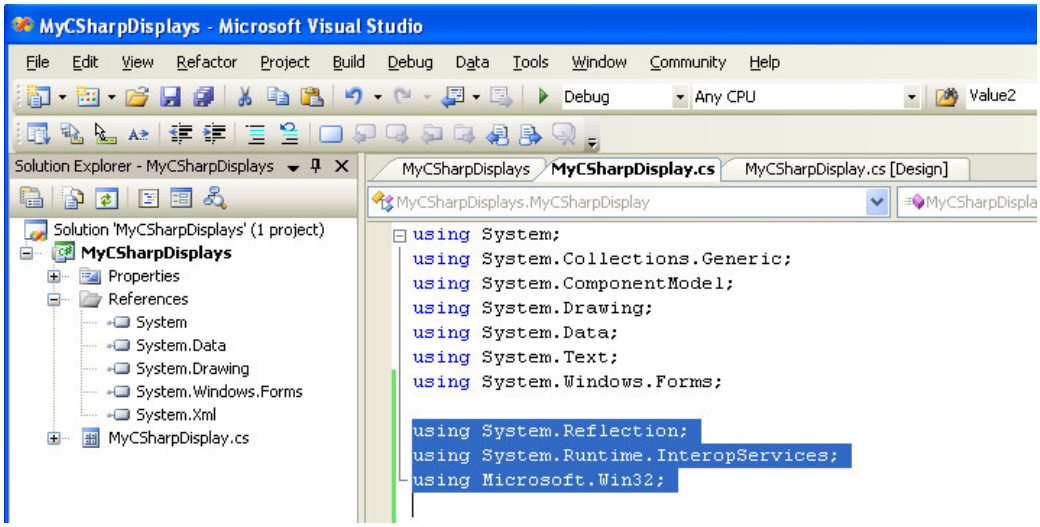


8) Now that we are done adding components, we will need to do some work in the code section. Go to the Solution Explorer and right click on your display.cs file. Choose “View Code”.



9) In your display.cs file, add the following “using” clauses:

```
using System.Reflection;
using System.Runtime.InteropServices;
using Microsoft.Win32;
```

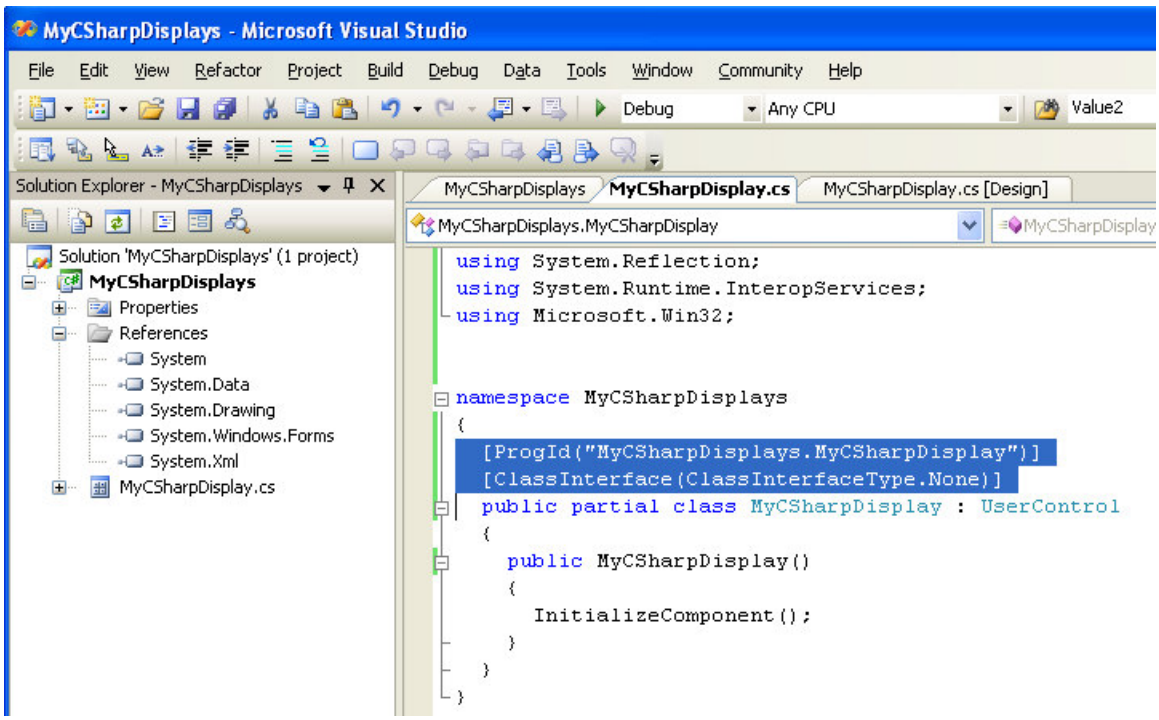


10) Now, add the following attributes to your class:

```
[ProgId("MyCSharpDisplays.MyCSharpDisplay")]
[ClassInterface(ClassInterfaceType.None)]
```

Note that the ProgId attribute is the same “ProgID” as mentioned above in step 3. This will become your display’s name inside of Iads, so choose appropriately. If you’ve followed the steps, the name should be in the form of ProjectName.ClassName.

There are several ClassInterfaceType options. "None" provides the fewest default properties beyond the ones you explicitly specify.

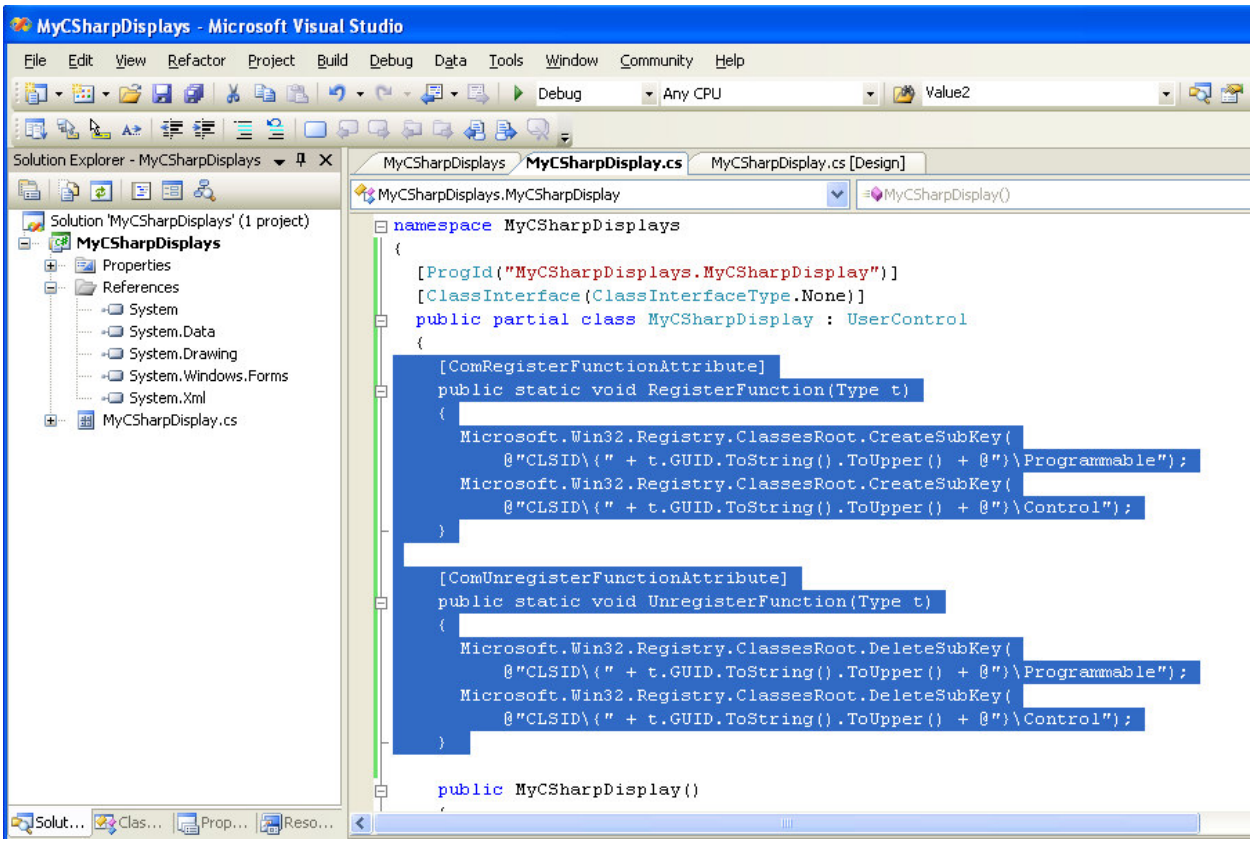


11) Add the following methods to perform the COM registration functions. COM is the interface method that Iads will use to send/receive data, save, and load your display. The registration mechanism is the manner in which Iads will identify your display after it is installed on a PC. If you skip this step, you will not be able to see your display in the Iads Display Builder dialog.

```
[ComRegisterFunctionAttribute]
public static void RegisterFunction(Type t)
{
    Microsoft.Win32.Registry.ClassesRoot.CreateSubKey(
        @"CLSID\{" + t.GUID.ToString().ToUpper() +
@"}\Programmable");
```

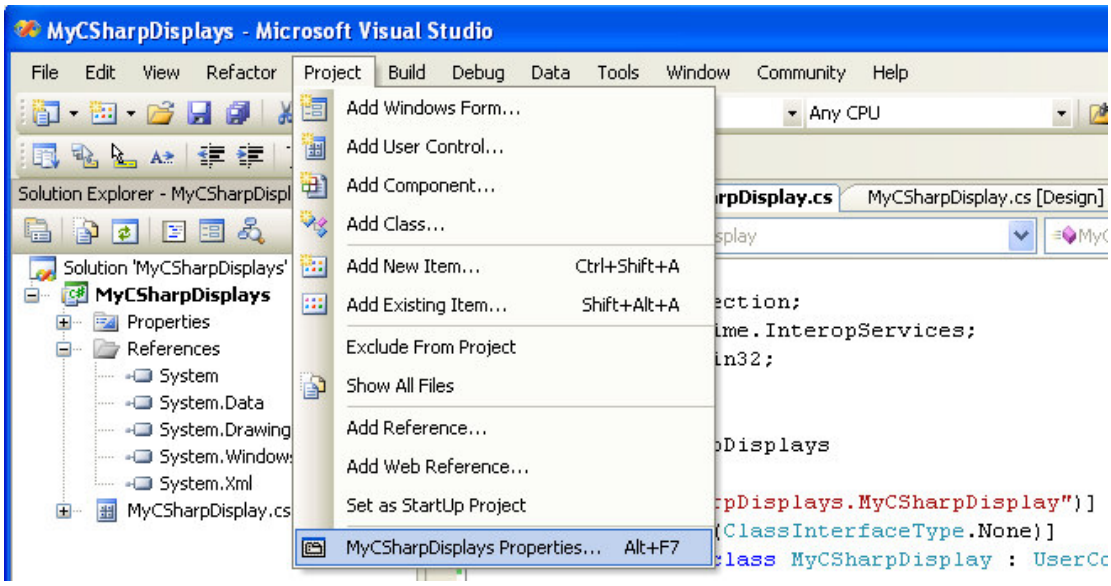
```
    Microsoft.Win32.Registry.ClassesRoot.CreateSubKey(
        @"CLSID\{" + t.GUID.ToString().ToUpper() +
@"}\Control");
}

[ComUnregisterFunctionAttribute]
public static void UnregisterFunction(Type t)
{
    Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey(
        @"CLSID\{" + t.GUID.ToString().ToUpper() +
@"}\Programmable");
    Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey(
        @"CLSID\{" + t.GUID.ToString().ToUpper() +
@"}\Control");
}
```

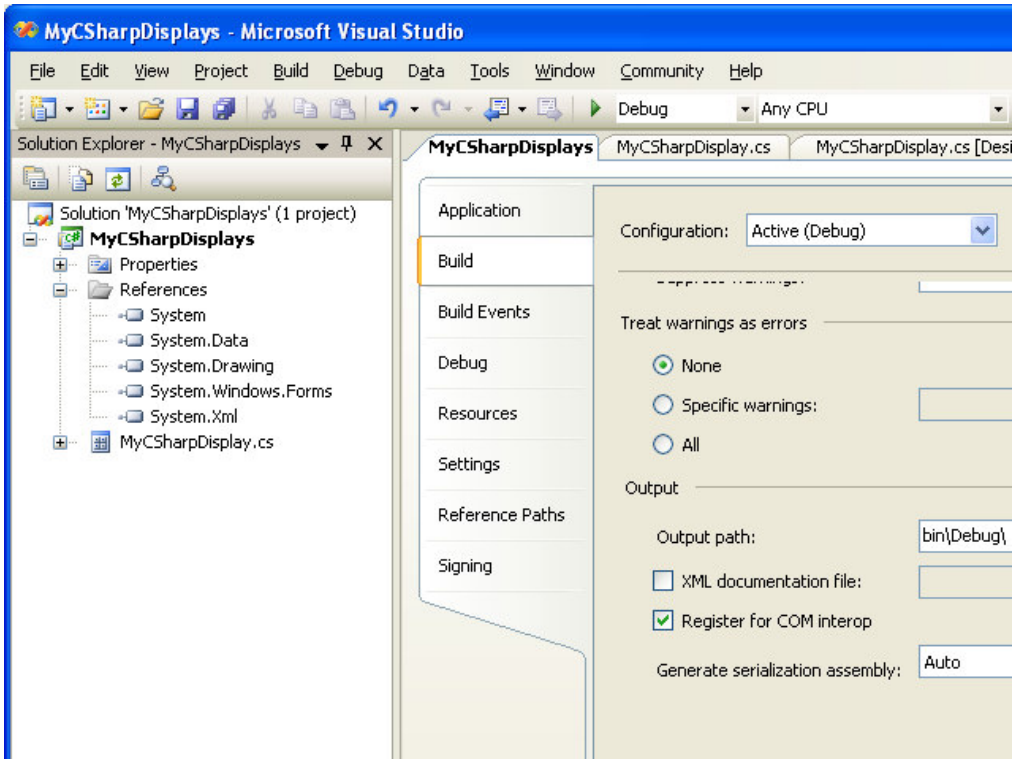


The RegisterFunction is called when the display is registered during the execution of the RegAsm utility (as discussed above). The UnregisterFunction is called when a display is removed from a PC.

12) Now, we must ensure that the display is compiled with the necessary COM code so that it can communicate with Iads. In the “Project” drop down menu, select “Properties".



13) Under the “Build” tab, scroll down to the bottom and check the “Register for COM interop” option.



14) After this step in complete, save all of your work and continue on to the next section.

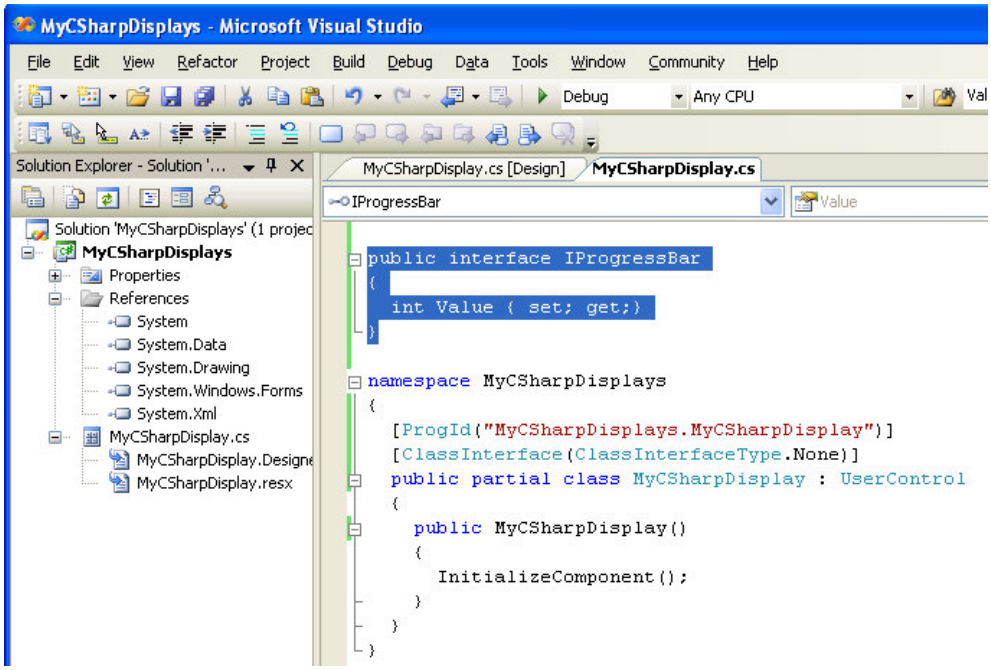
2.1. Adding Properties to the Display

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, imagine the "Attitude Indicator" display within Iads that simulates an aircraft dial. It has properties for "Roll", "Pitch", and "Heading" as well as "SkyColor" and "GroundColor". Changing the "Pitch" property in the attitude indicator would, as expected, cause the display to rotate its graphics to indicate the new pitch angle.

Any property that you include in your display will be an "access point" in which the user can modify its contents/characteristics/behavior. The magic of building an custom display 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) Add a public interface to the class. This is the interface in which we will put all of our properties that we want to expose to the user (as thus Iads). In our example, we have added the Value property to provide access to the Value property of the .Net ProgressBar control.

```
public interface IProgressBar
{
    int Value {set; get;}
}
```

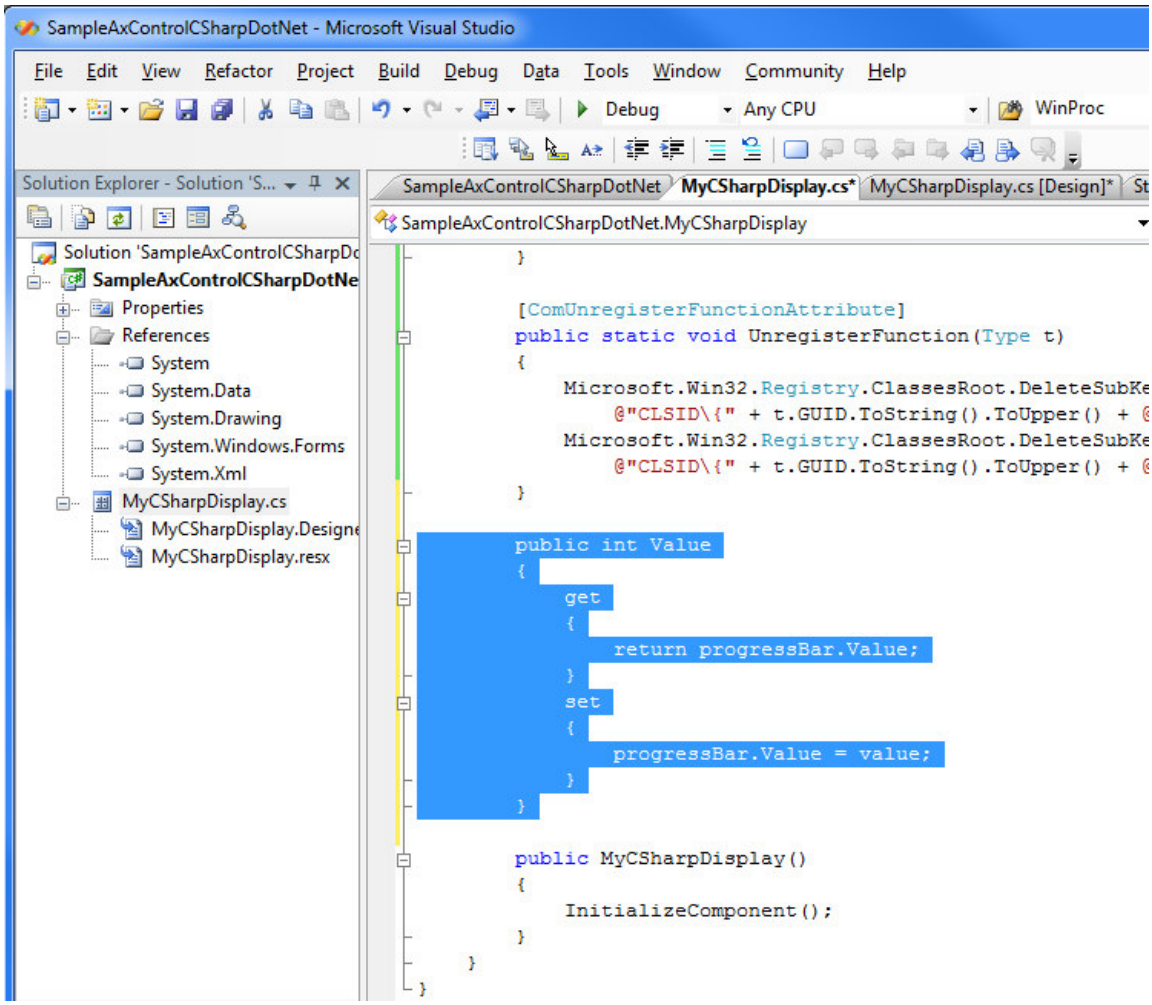


2) Add the interface to the User Control class:

```
public class ActiveXDotNetControl :
System.Windows.Forms.UserControl, IProgressBar
```

3) The next step is to implement the code created for each new property we added. Thus, in our example property "Value", we will need to focus in on the set and get functions.

4) For the set function, we will need to capture the incoming value and push it into the progress bar object. Once that is complete, our progress bar should redraw and display the new value. Please be aware that the value from the outside might be pushed on this display at a very high frequency. It might be prudent to add code to determine if the incoming property value is different than the existing value of the property and forgo the setting on the progress bar. This is a vital optimization you might need to consider when building more complex displays. For this simple example, we will skip this step. The get function is easy to implement. Simply return the value of our progress bar that we have created for this property.




```
public int Value
{
    get
    {
        return progressBar.Value;
    }
    set
    {
        progressBar.Value = value;
    }
}
```

5) 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. Continue on to the next section if you want to learn how to view and debug your display from within Iads.

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

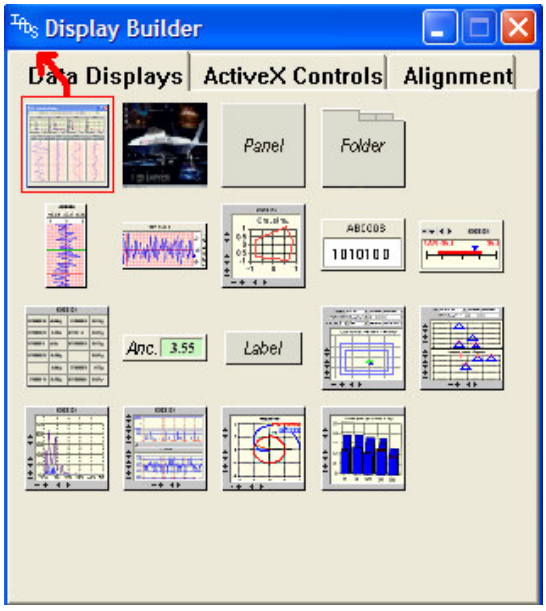
If you have any further questions, you can search the Iads Google Group or post a question:

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

3. Adding Your New Control to IADS

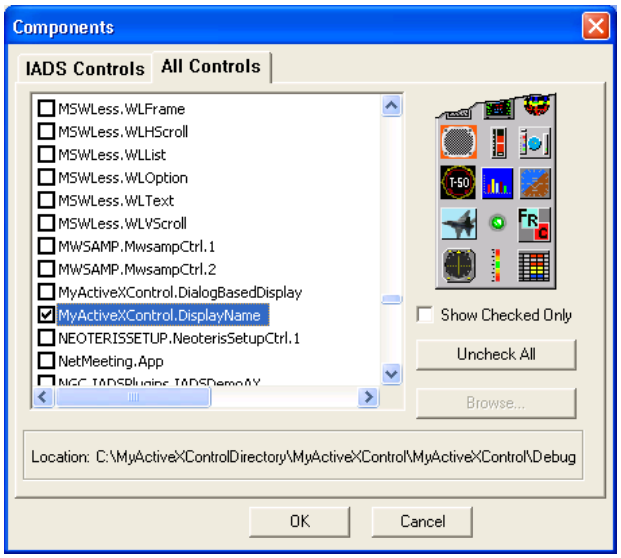
1) Press the "Display Builder" button on the IADS "Dashboard" in the lower right hand corner of the screen. The "Display Builder" dialog will appear with icons of components that you can use to build your displays (including your new ActiveX control).

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 "Program ID" as discussed earlier in this tutorial. In the example code, my control is named "`MyCSharpDisplays.MyCSharpDisplay`". 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.

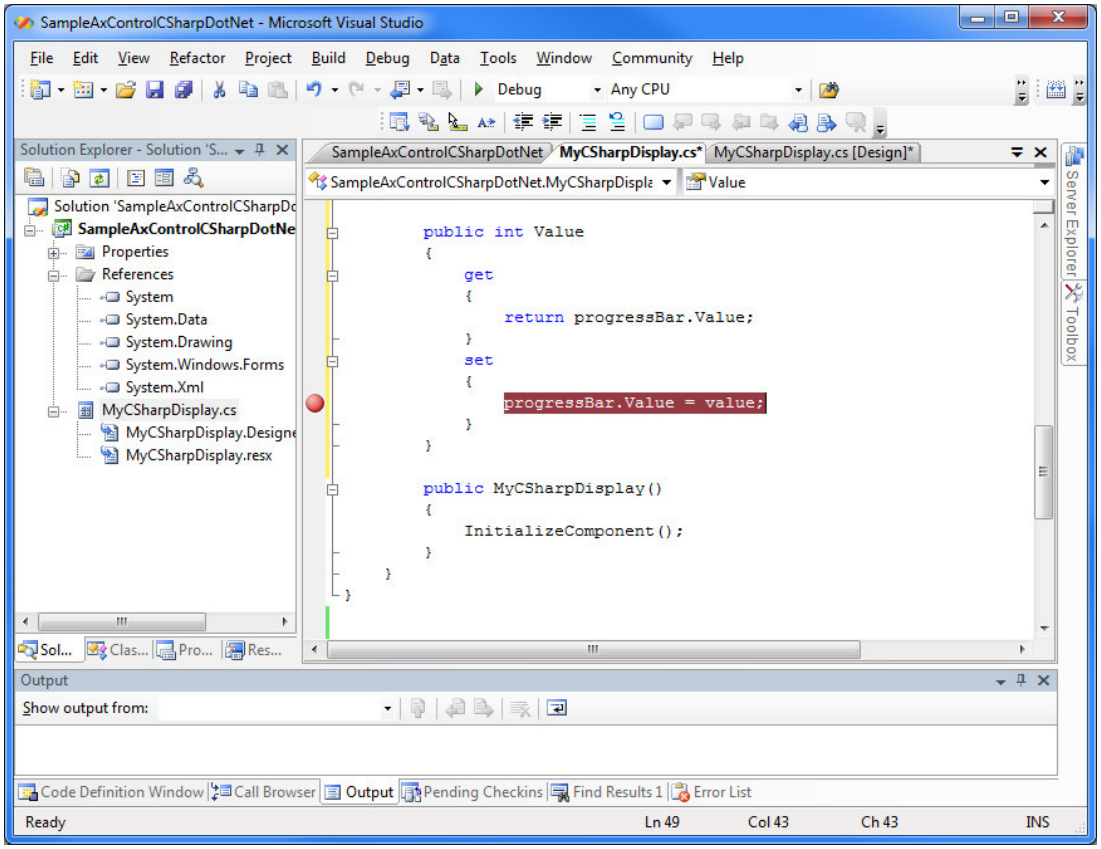
Note: If you can't locate your display in the "All Controls" list, try checking your ".cs" file in. It contains a `ProgId` section. This is where the "ProgId" (Program ID) is designated for the specific object you created within your project.



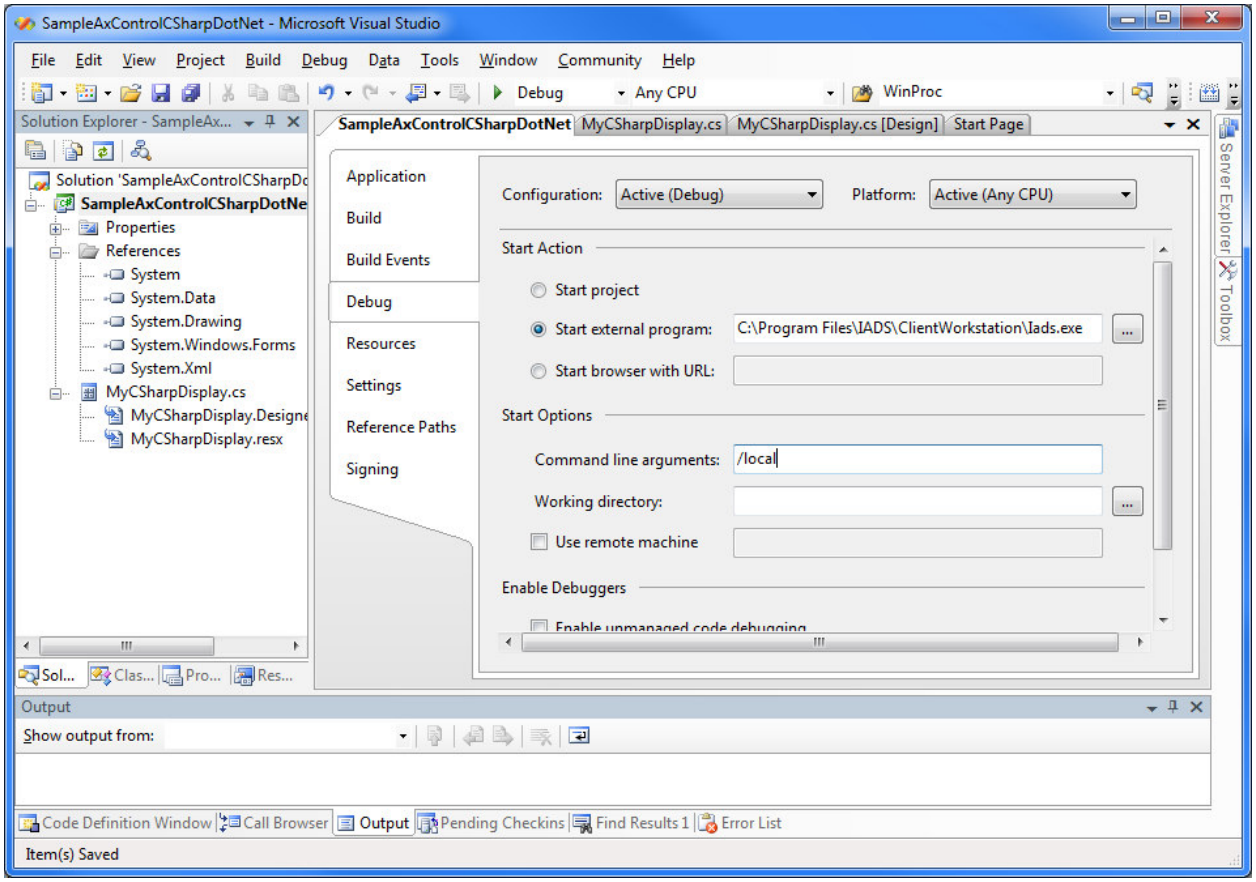
4) Once you've added your display to the Display Builder tool you can simply drag on drop the icon into the Analysis Window you've created in step 2. At this point, your display will be created. In many instances, you will have a problem that needs to be addressed. In this case, continue to the next section to learn how to debug your display.

4. Debugging Your New Control in IADS

1) In the development environment, place a break point in one of the "Set" method for testing.



2) In Visual Studio, select the “Project->Properties” drop down menu. In the “Debug” tab, set the “Start external program” field to the Iads application executable (Iads.exe). The exe is located in the “C:\Program Files\Iads” directory. Also, set the “Command line arguments” field to “/local”. 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. Once that is complete. Save your configuration. Choose a parameter from the parameter tool and drop it into the display. After the parameter is attached, your break point should now hit in the debugger. You can now step through your drawing code if necessary.