Creating an IADS Custom Function Using C# VS2005

January 2013

SYMVIONICS Document SSD-IADS-049, Rev 8.0

SYMVIONICS, Inc.
Telemetry Systems

Created November 20, 2012

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:

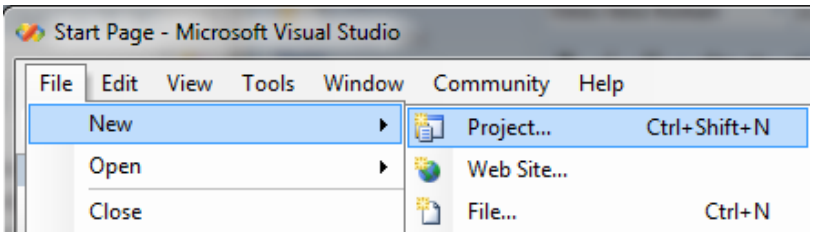
1) Creating your new function using the VS2005 C# Project Wizard

2) How to access your new function in IADS

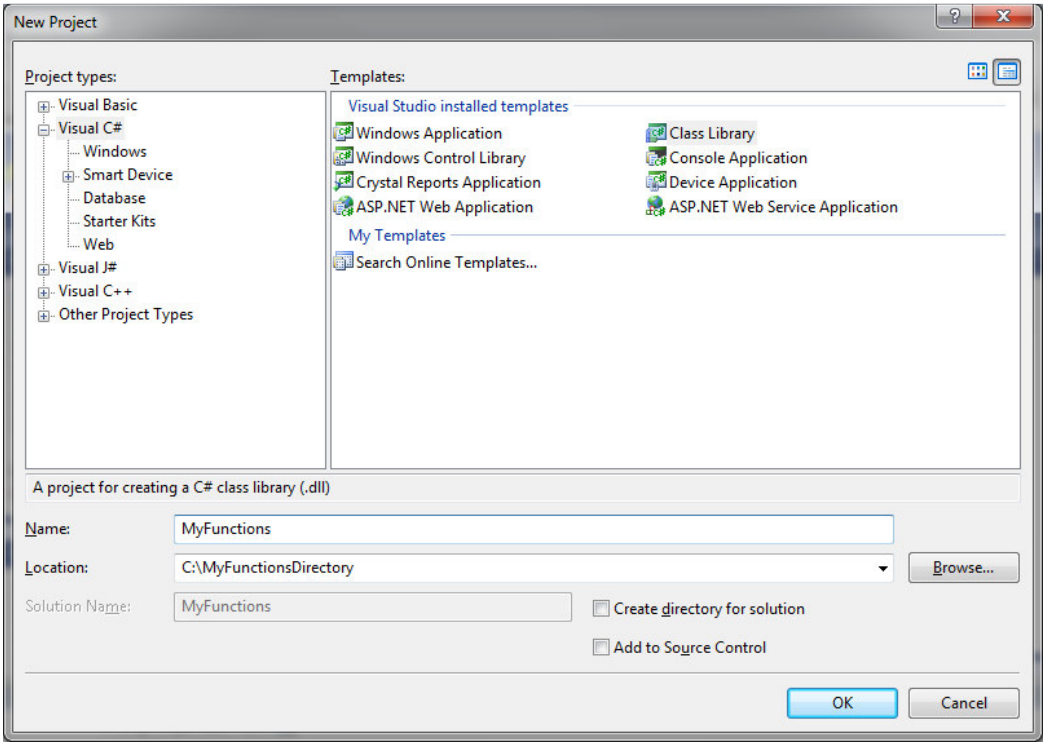
3) How to debug your new function in IADS

2. Creating your new function using the C# Project Wizard

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



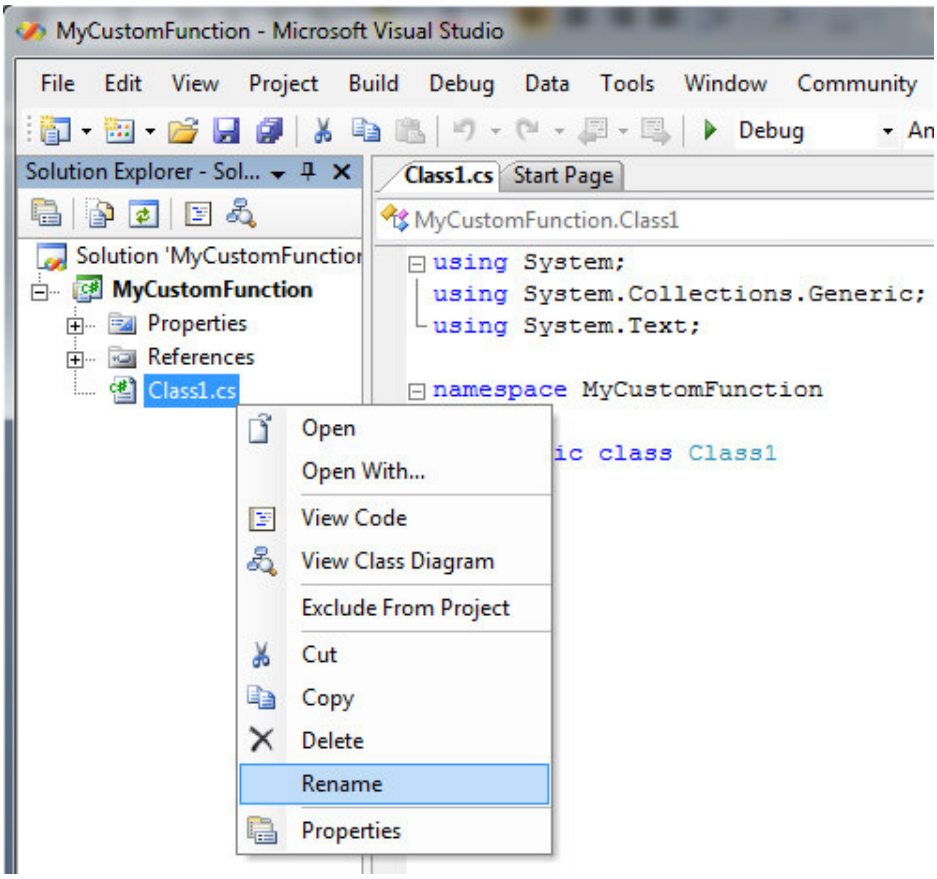
2) In the New Project dialog that appears, choose the “Visual C#” tier and click the Class Library” option. At this point, please read the next step before you finish completing the dialog. There are some important considerations when choosing the proper project name.



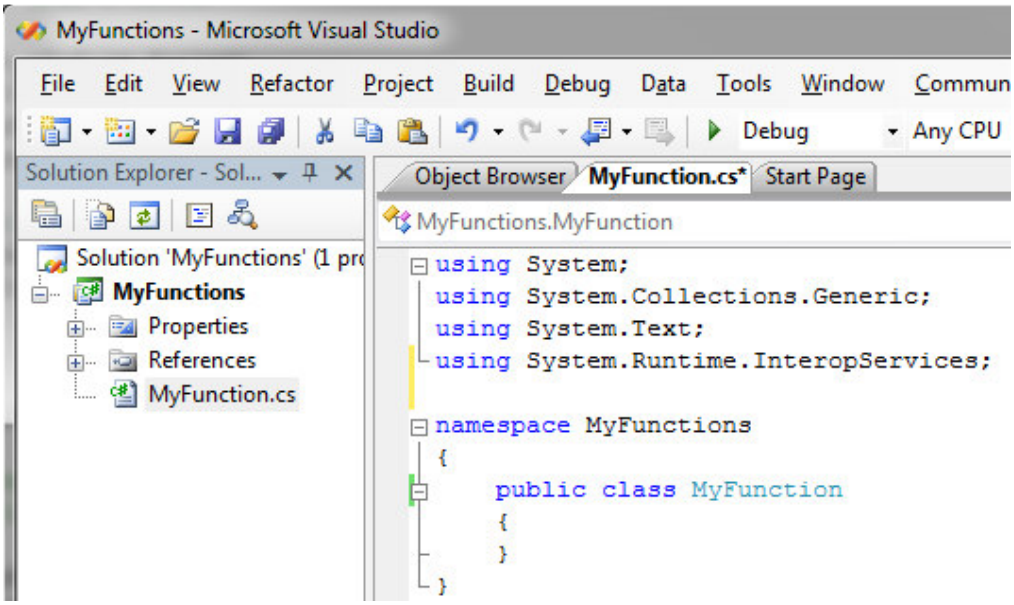
3) Plan on creating many functions in one “project” (most common and easier to manage the code). One way to look at it is that the project name is akin to the “Genus” of your function, so shoot for generality. Consider prefixing the project name with your organization like “Nasa” or “Lockheed” and the type of functions you’ll be adding (example: NasaFluidFuncs).

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

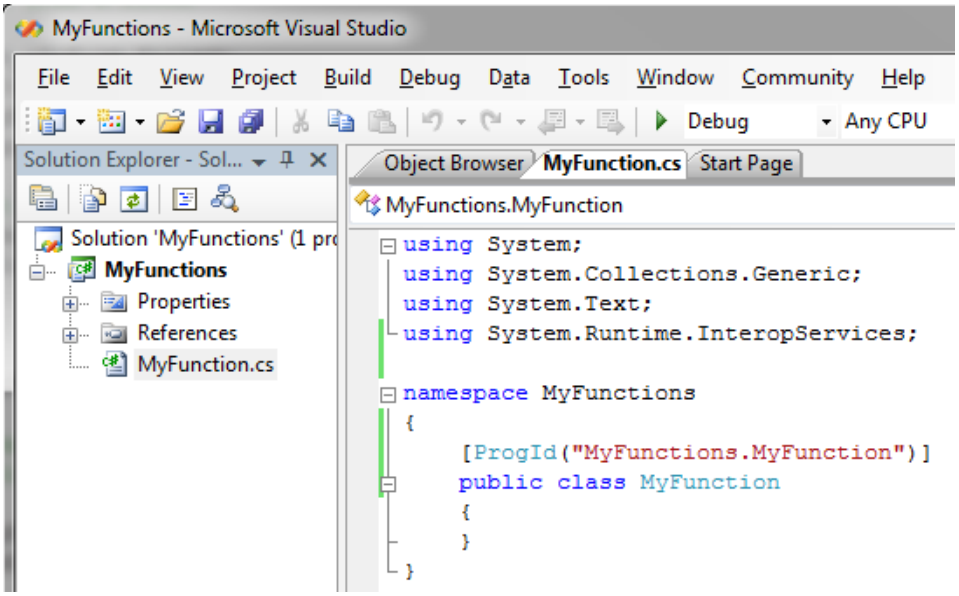
4) After pressing OK, the project will be ready for editing. Before we begin, rename the Class1.cs file to reflect our function name. In this example, we’ll call it ‘MyFunction.cs’.



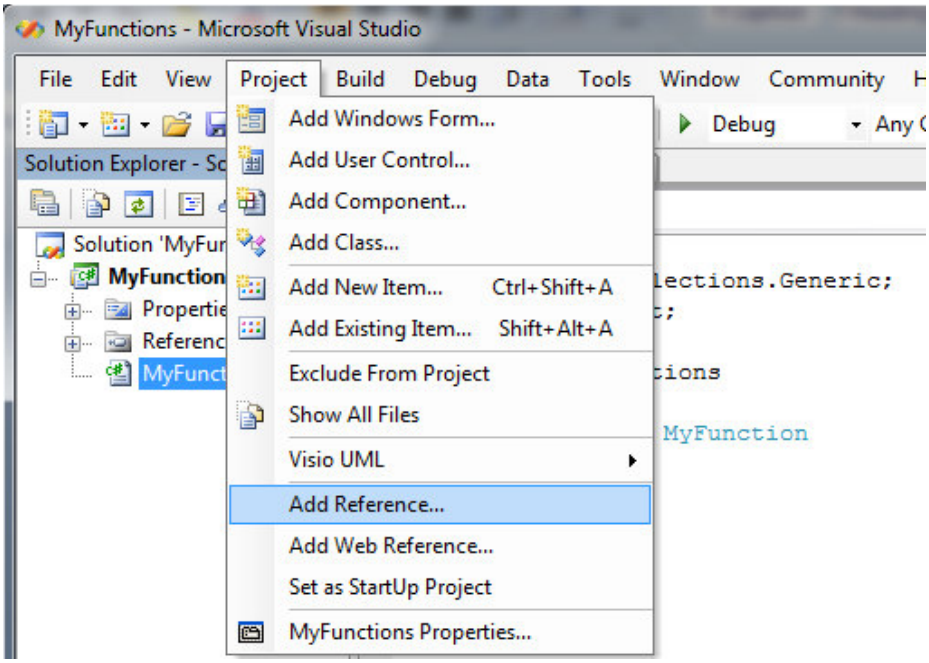
5) In preparation for the next step, we'll need to add another "using" directive. In the code view, add a line "using System.Runtime.IteropServices".



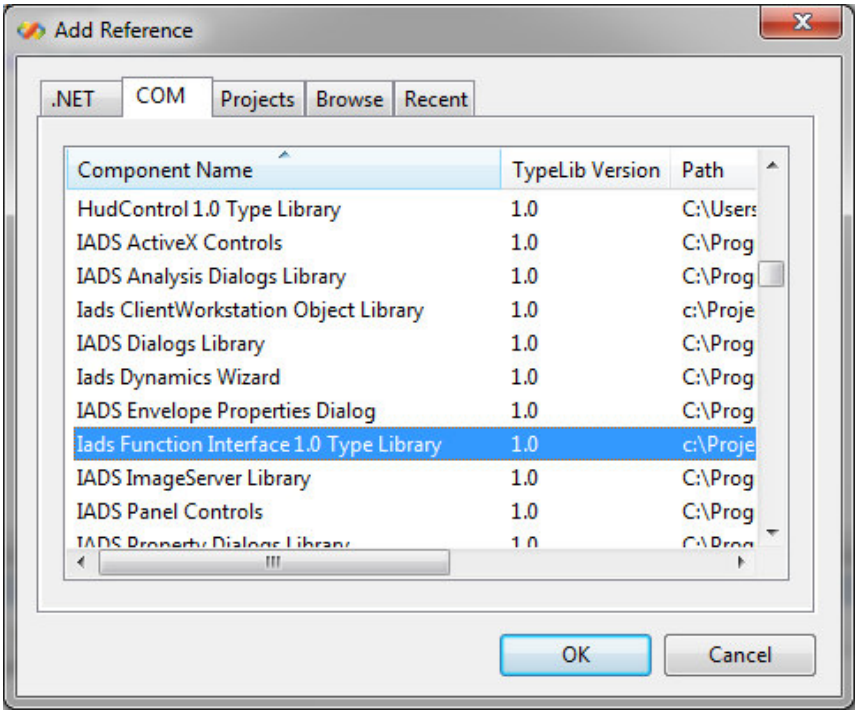
6) Now we need to focus on the entire function name as it will appear to the Iads end user. The format of the function name requires two strings separated by a period ('.'). It's best practice to use the Visual Studio project name as the first portion of the name (before the period). The portion after the period should be your specific function name. For instance, ProjectName.ClassName, NasaFluidFuncs.FlowRate, or in this tutorial MyFunctions.MyFunction. We will define that name explicitly by using the "ProgId" directive. In the code view, type [ProgId("MyFunctions.MyFunction")] above your class definition "public class MyFunction".



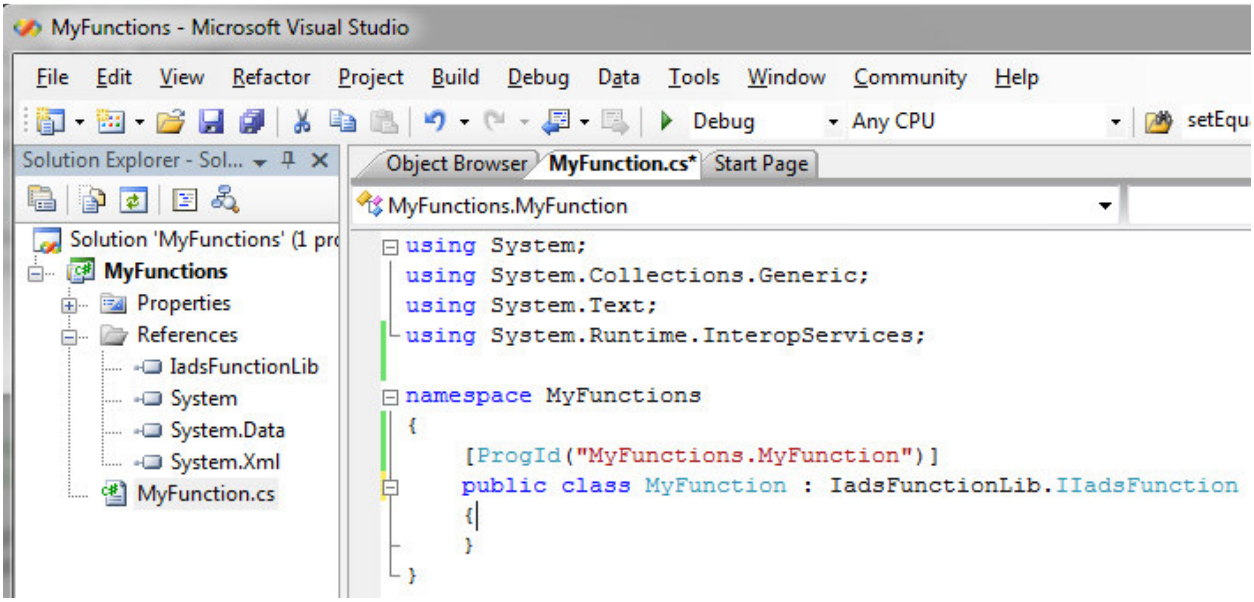
7) The next step is to implement the interface that Iads requires to call your function. To accomplish this task, we'll need to add a reference to the definition file of the interface. From the Project menu, select "Add Reference".



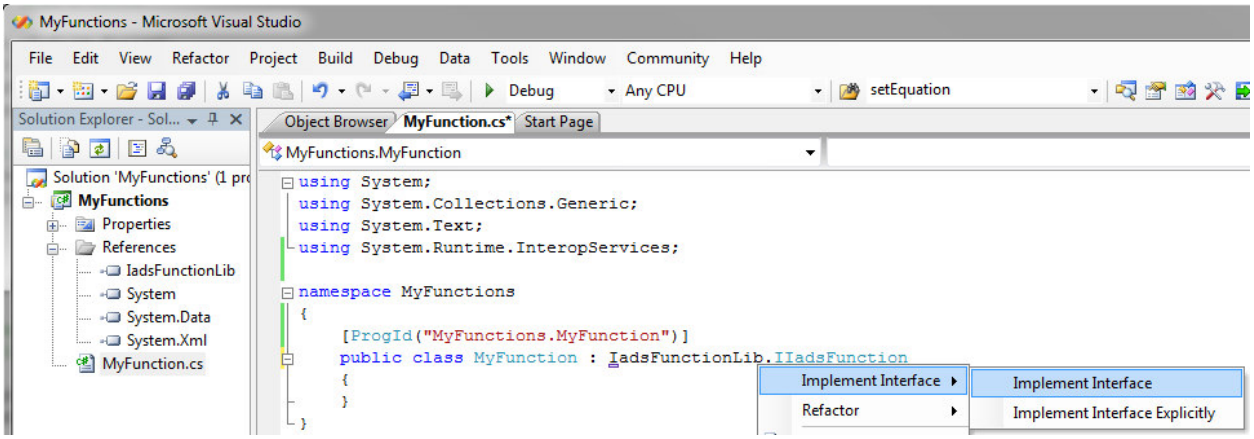
8) In the "Add Reference" dialog, select the "COM" tab. Scroll down until you see "Iads Function Interface 1.0 Type Library". Press Ok to add the reference to our project.



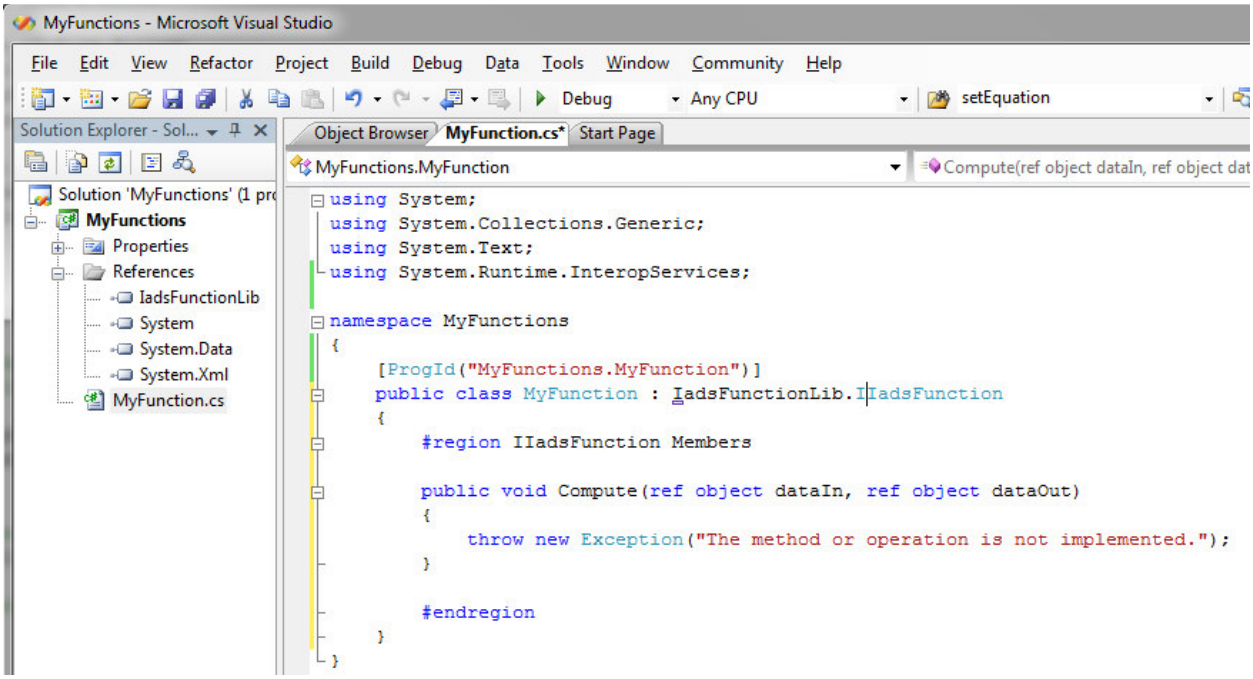
9) Now, back in the Solution view, notice that the IadsFunctionLib has been added to our References section. This reference contains the definition of the Iads custom function interface. So far so good. Now, we'll need to add the interface to our class definition. First, type a colon ':' after the class name and add the text "IadsFunctionLib.IIadsFunction". In essence, we'll inherit from the interface definition. After this is complete, we can implement the interface.



10) To implement the interface, right click on the "IadsFunctionLib.IIadsFunction" text in the code view window. In the popup menu, select "Implement Interface->Implement Interface".



11) After choosing the “Implement Interface” menu item, Visual Studio automatically writes the shell of the function we must implement. Within the function, the “ref object dataIn” argument represents an array of input argument values from the Iads environment. The “ref object dataOut” represents the single return value that we are allowed to return. For example, if a user typed a derived equation MyFunctions.MyFunction(1, 2, Param1), the dataIn object would be an array of three element containing the values 1, 2, and the value of Param1 respectively. Our job is then to take the input values, run some mathematical algorithm, and produce a single output ‘result’. Discussions on returning multiple results from a single function call will be touched upon at a later time. For simplicity sake, let’s just focus on the multiple in, single out methodology.



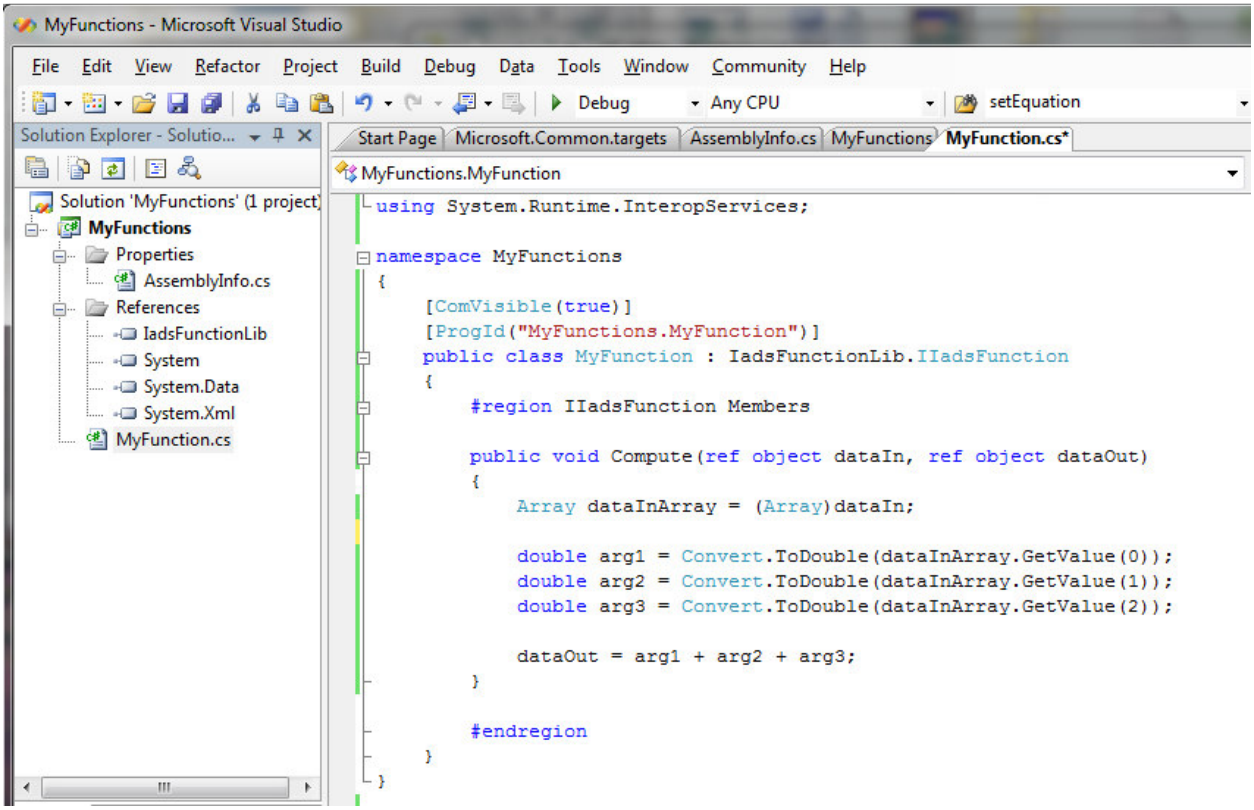
12) During this step, we’ll need to decode the dataIn object and extract the input parameter values. Once that is complete, we can perform our calculation and return the result. In the code window, remove the line of code containing the “throw” statement. Now, add the following code in its place:

```
Array dataInArray = (Array)dataIn;

double arg1 = Convert.ToDouble(dataInArray.GetValue(0));
double arg2 = Convert.ToDouble(dataInArray.GetValue(1));
double arg3 = Convert.ToDouble(dataInArray.GetValue(2));

dataOut = arg1 + arg2 + arg3;
```

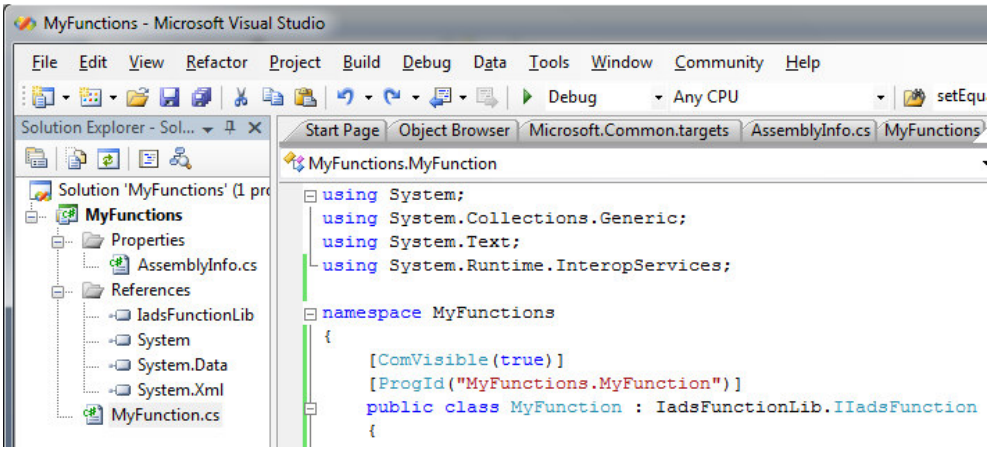
When you are complete, your code should appear as follows:



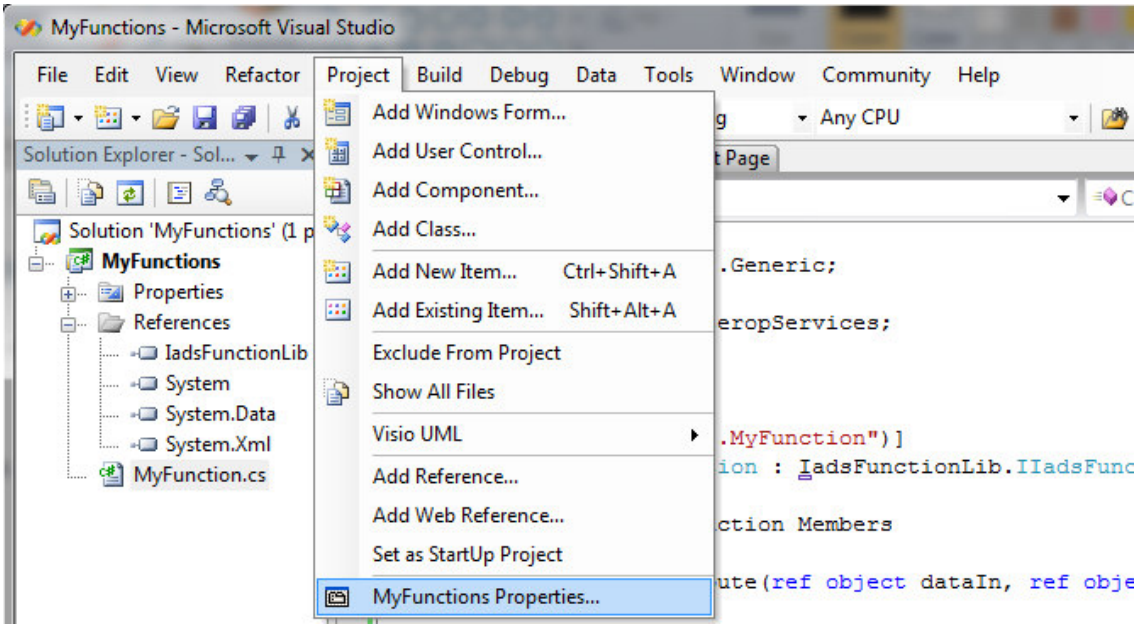
13) Notice in the first line, we cast the 'dataIn' array to a C# Array type object. This will allow us to extract each function input argument in the subsequent lines of code. The first argument passed into this function from Iads is array element 0, the second argument is 1, and so on. By using the 'Convert' object, we can assign the each input argument to a temporary variable. Once these temporary variables are assigned, we can perform our calculation and return our result. To a return a result, simply assign the computed value to the 'dataOut' object.

14) At this point in time, you can modify the code to suite your exact calculation. The example code shows how to compute the sum of three input arguments. Simply modify the code to include your algorithm and the desired number of input arguments.

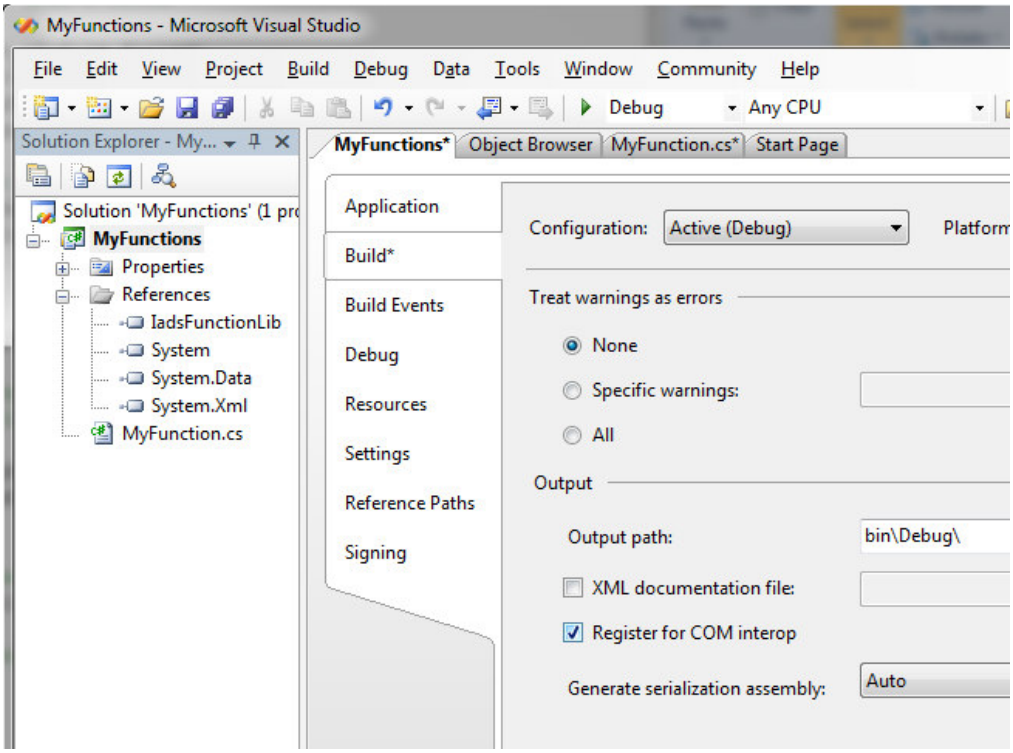
15) Now that your function is basically complete, we must take care of some remaining interop issues. We must ensure that the function is compiled with the necessary COM code so that it can communicate with Iads. In the code view, type `[ComVisible(true)]` above the ProgId definition line.



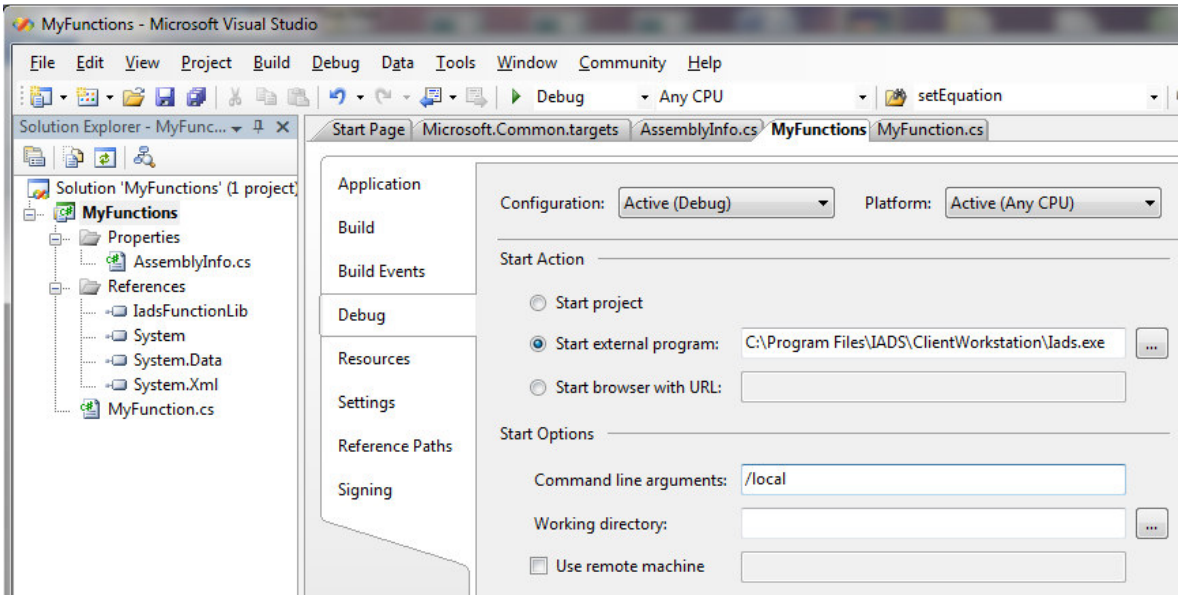
In the “Project” drop down menu, select “Properties".



16) Under the “Build” tab, scroll down to the bottom and check the “Register for COM interop” option. These two last steps are very important. If you forget this step, or the previous `[ComVisible(true)]` directive step, the function will remain undefined in Iads because it will not be properly registered on the system.



To debug the function, go to the “Debug” tab in the same dialog and set the “Start external program” item to the location of the Iads executable, in this case “C:\Program Files\IADS\ClientWorkstation\Iads.exe”. In Start Options, set the “Command line arguments” to “/local”.



17) When all these steps are complete, compile the project, fix any errors and run. In Iads, build a new derived parameter that calls your new function, and drop the parameter in any display. If you want to debug your calculation step by step, put a break point in your compute function. The code will break for each new data point calculated. If you need more specific information on these steps, continue on to the next two sections.

    For more background on how to build a custom function, see the sample C# project including with this tutorial and read the comments within the code. Continue on to the next section if you want to learn how to view and debug your function from within Iads.

    http://iads.symvionics.com/programs.html

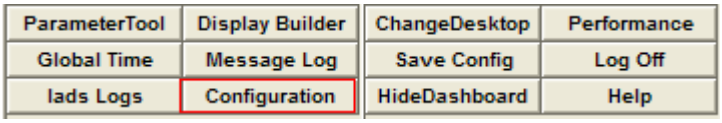
    If you have any further questions, you can search the Iads Google Group or post a question:

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

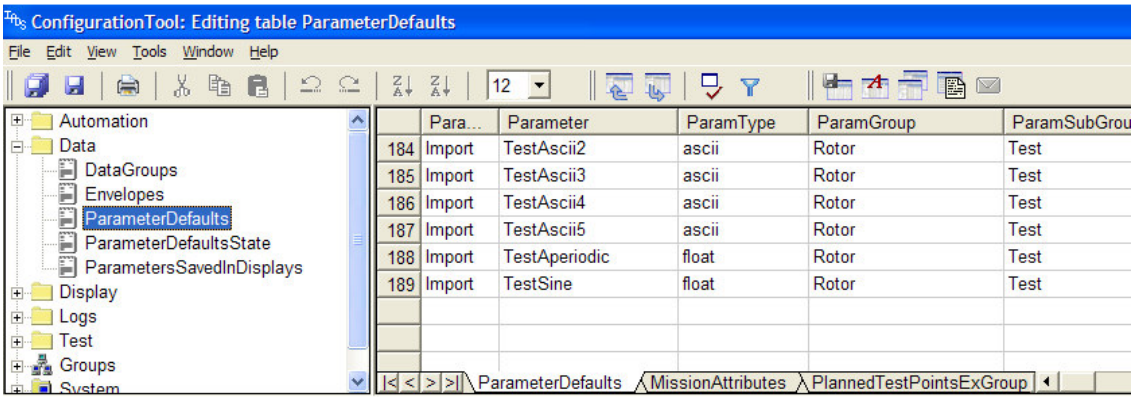
3. Accessing your new function in IADS

1) Run IADS and login to a test Desktop.

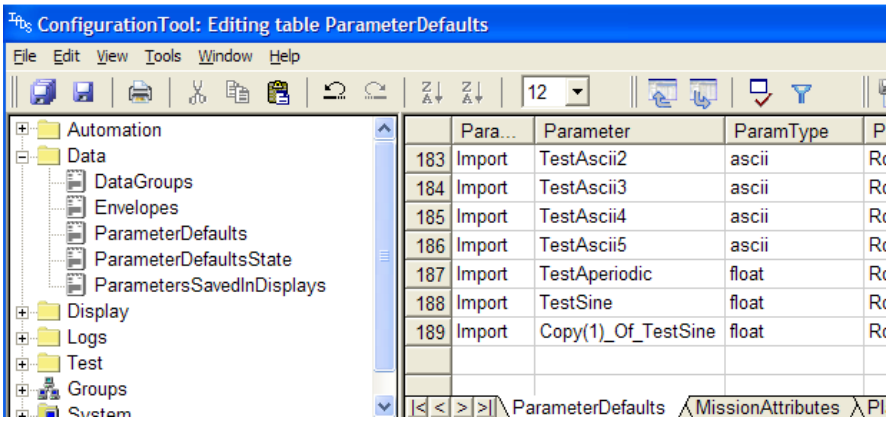
2) Click the "Configuration" button on the IADS Dashboard in the lower right corner of the screen.



3) After pressing the button, the Configuration Tool dialog will appear. In the left window pane, click the "Data" folder and then finally the "ParameterDefaults" table. This is the location in Iads where you will build a new derived parameter to test your function.



4) Ok, let's add a new derived parameter. For speed, we'll simply copy the last line in the table and then replace our new values as needed. Select the last row in the table by pressing the row button (in the picture, row #189). After the row is selected, press <Ctrl+C> to copy and then follow that by a <Ctrl+V> to paste. You should now see a copy of the last line placed into a new row. When you're done, the table should look something like this:



5) Click into the first column of the new row. As we go along, to proceed to the next cell simply press the "Tab" key.

   Leave the first column alone and simply press the Tab key to start editing the second column. In the second column, type the name of your test parameter. Let's call it "TestMyFunction". Once you are done, press the Tab key as always. Now let's set the type of the parameter. Just leave it "float" (i.e. 4 byte floating point number). In the future, if you're testing an Ascii return value, you'll need to set this to Ascii.

   At this point, keep pressing the Tab key until your arrive at the "DataSourceType" column. Make sure that is set to "Derived".

   In the next column (DataSourceArgument) you'll write your derived equation. Now you will use the knowledge learned from the discussion in the above tutorial regarding the project name and function name. Enter the function name followed by the arguments:

   MyProjectName.FunctionName(5.0, 10.0, 30.0)

   If you've used the same names as the tutorial above, the function would we written like this:

   MyFunctions.MyFunction(5.0, 10.0, 30.0)

   If you want some variety to your test data, you can use something like this:

   MyFunctions.MyFunction (Rand()*5.0, Rand()*10.0, Rand()*30.0)

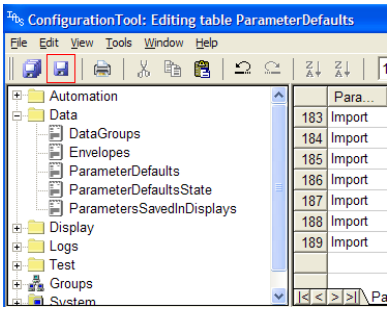
   Or if you already have specific input parameters in mind, you can do something like this:

   MyFunctions.MyFunction (Param1, Param2, Param3)

   In the next field (UpdateRate), type the sample rate that you desire to update your function. If your equation is based off of other parameters, the sample rate will be automatically computed and placed into this field when you Tab out of the cell.

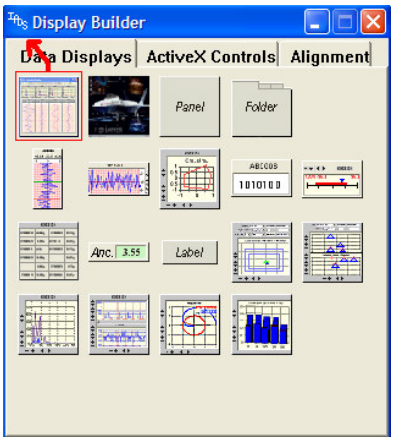
   Just for safe measure, press the Tab key until you get to the "FilterActive" column. Make sure that it is set to "No". We don't want a filter to be affecting our output at this time, or it could lead to confusion.

   After these steps are complete, press the "Save" toolbar icon in the Configuration Tool; your new parameter will appear in the Parameter Tool.

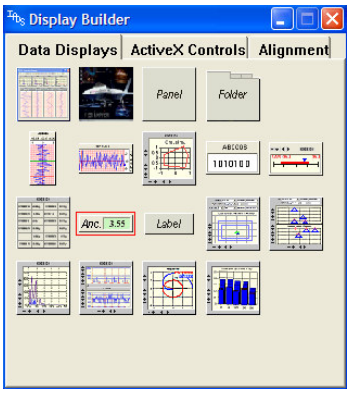


6) To run the function, simply drop the parameter into any display. If you're not familiar with building a test display and attaching a parameter, continue the tutorial for more instruction.

7) To build a test display, simply create an empty Analysis Window by dragging the icon from your Display Builder tool and on to your Microsoft Windows desktop and dropping it. After you've dropped the Analysis Window, you have a choice to name the window.

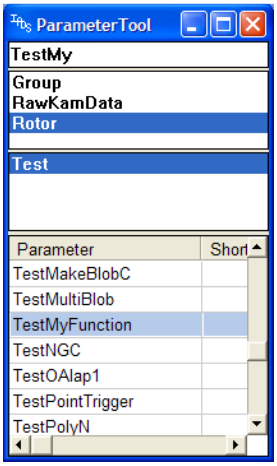


8) Now simply repeat the process, but drop the "AlphaNumeric" display into the Analysis Window (be sure drop the new display into the Analysis Window you've just created and not on to the Microsoft desktop). After the drop is complete, you should see the new display in the Analysis Window. The AlphaNumeric is a very simple text display that will be easy to view our equation output results.



9) Ok, now for the parameter attachment to the display. Click on the "Parameter Tool" button in the Iads Dashboard (bottom right hand corner of screen). The Parameter Tool dialog will appear. The Parameter Tool dialog contains a list of all your available parameters in the configuration. Now all we need to do is find our parameter.

10) In the top text field (quick find box), start typing the parameter name. I used the name "TestMyParameter", so if you've done the same then simply type "TestMy". You'll notice that the window at the bottom opens as soon as it finds your parameter. Keep typing until you see the full parameter appears. Once it's visible, click on the parameter name and "drag" the parameter into the display on the Analysis Window. As soon as you drop the parameter, data should appear. This is the actual output of your function! See that wasn't too bad ;)

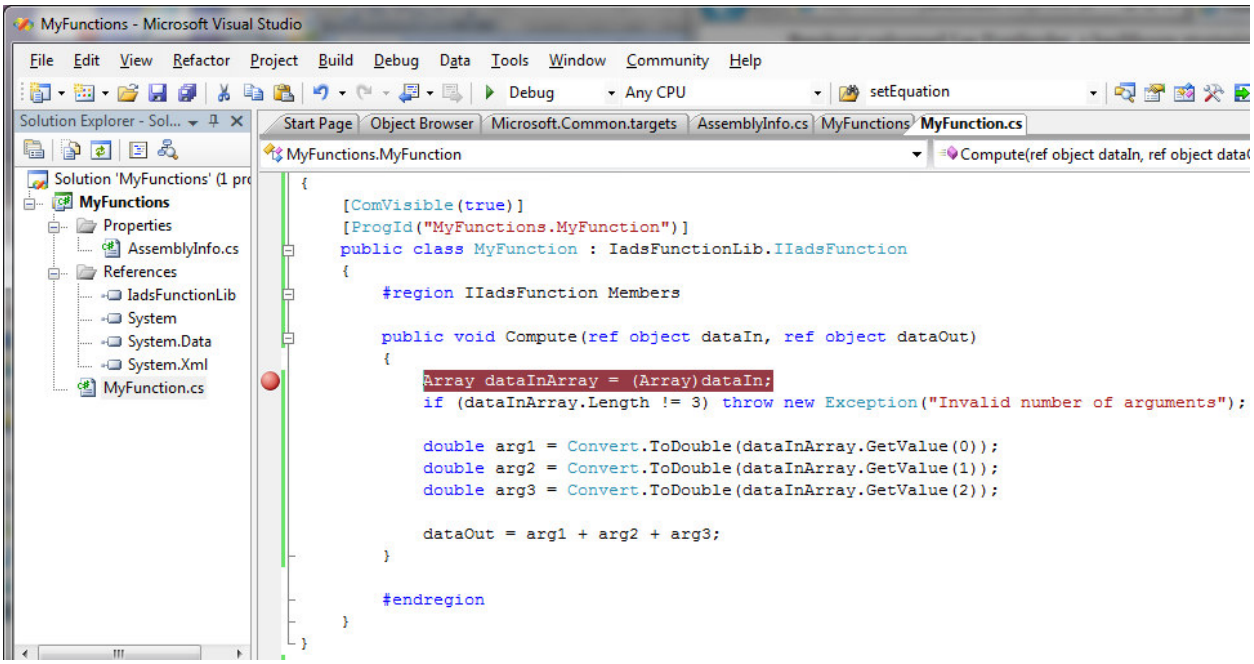


11) After your initial checkout is complete, you can move on to displays such as the Stripchart that will show history and allow you to examine the data point by point for discrepancies. Simply repeat the process above at step 8, but in this instance use the icon just under the Analysis Window icon (first column second row). Make sure to save the configuration for later.

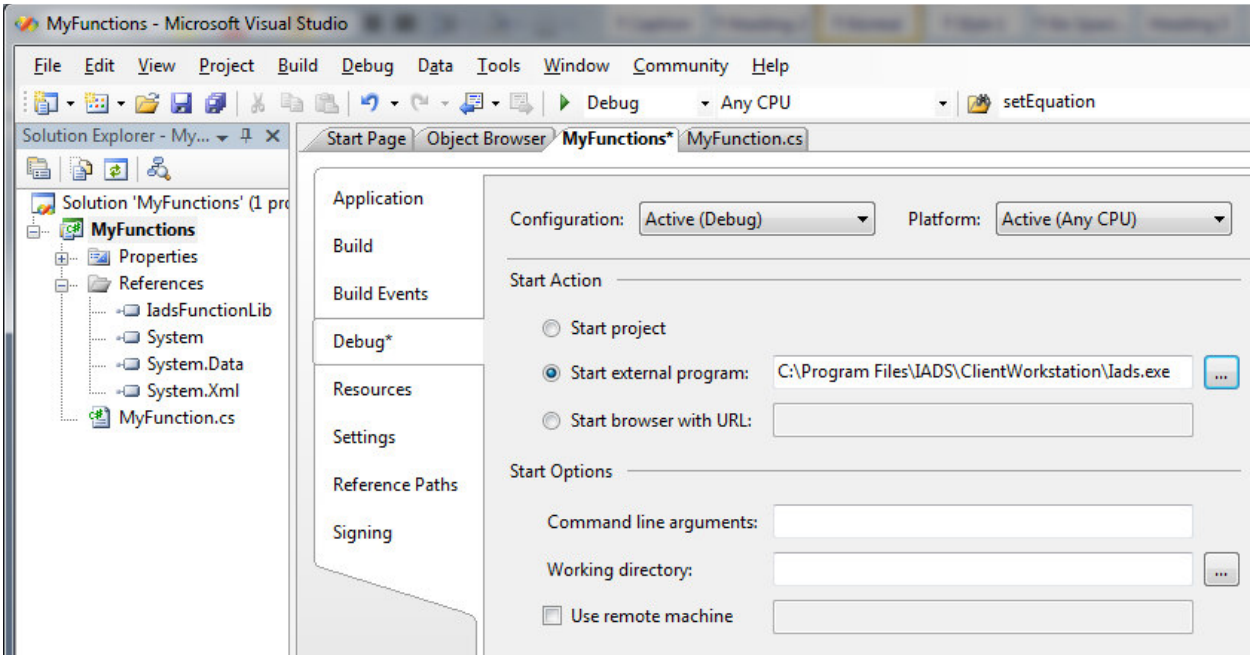
    If you want to debug the function using the Visual Studio 2005 debugger, continue to the next section. This is the quickest way to ensure that your function is operating properly and you are highly encourage to run through you function at least once using the debugger.

4. Debugging your new function in IADS

1) Bring up your Visual Studio custom function project, and place a break point in your "Compute" method for testing.



2) To run Iads from the debugger, go to the "Debug" tab in the same dialog and set the "Start external program" item to the location of the Iads executable, in this case "C:\Program Files\IADS\ClientWorkstation\Iads.exe"



3) Build your Solution again for good measure and click on the "Start Debugging" command (or the F5 key). Iads will start. When Iads starts, pick the configuration file you wish to use (possibly the same configuration file as used in the last section) and click Open.

4) After Iads initializes, open up the Configuration Tool and create a derived parameter in the ParameterDefaults table. If you need more background info on how to do this, consult the last section. If you've already created a derived parameter referencing your function, simply click on your equation in the ParameterDefaults table.

   Notice that when you "tab out" or finish the equation in the ParameterDefaults table, your function will be called. At this point you can debug all of the argument types and make sure you're getting the correct items. If you have an argument error and return an error code from your function, notice that you'll get an error message inside of Iads and the equation text will turn red in color. Once you've checked out the arguments, you can remove the breakpoint and debug the function with live data.

5) Add a display to the new Analysis Window (i.e. AlphaNumeric or Stripchart) as described in the last section. If your parameter isn't already attached to a display, simply drag and drop your newly built derived parameter into the display. Your break point should now hit in the debugger. You can now step through your computational code if necessary.

   Again, for more background on how to pass arguments, check their types, and return values, please read the comments in the supplied example project

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

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