Fusion API Training

User Guide

# Contents

## Table of Contents

[1 Contents 2](#_Toc410903422)

[1.1 Table of Contents 2](#_Toc410903423)

[1.2 Figures 2](#_Toc410903424)

[2 Topics covered 4](#_Toc410903425)

[3 Glossary 4](#_Toc410903426)

[4 Programs to Install 4](#_Toc410903427)

[4.1 C# Express 4](#_Toc410903428)

[4.2 Fusion API 2.0.7 December 23, 2014 4](#_Toc410903429)

[5 Part 1 – Getting Started 5](#_Toc410903430)

[5.1 Creating a “solution” 5](#_Toc410903431)

[5.2 COMPILE ALL APPLICATIONS FOR .NET 4.0 FRAMEWORK 7](#_Toc410903432)

[5.3 Hello World Console Program 9](#_Toc410903433)

[5.4 GUI Winforms Application 9](#_Toc410903434)

[5.5 My Library 16](#_Toc410903435)

[6 PART 2 – Fusion api 20](#_Toc410903436)

[6.1 Adding Fusion API library references 20](#_Toc410903437)

[6.2 smbus adapter 23](#_Toc410903438)

[6.3 Memory debugger – Memory maps! 29](#_Toc410903439)

[6.4 PMBus commands 32](#_Toc410903440)

[6.5 more examples - fusion api samples solution 33](#_Toc410903441)

[7 Advanced topics 34](#_Toc410903442)

## Figures

[Figure 1 – VS - File>New>Project 5](#_Toc410903443)

[Figure 2 - Express - New Project 5](#_Toc410903444)

[Figure 3 - VS - Console Application settings 6](#_Toc410903445)

[Figure 4 - Express - Console Application settings 6](#_Toc410903446)

[Figure 5 - VS - New Project created 7](#_Toc410903447)

[Figure 6 - VS - Add new Windows Forms Application 10](#_Toc410903448)

[Figure 7 - blank form 11](#_Toc410903449)

[Figure 8 - VS - Toolbox 11](#_Toc410903450)

[Figure 9 - Express - Toolbox 12](#_Toc410903451)

[Figure 10 - Pin the toolbox 12](#_Toc410903452)

[Figure 11 - Accessing the Properties window 13](#_Toc410903453)

[Figure 12 - Text property for label1 14](#_Toc410903454)

[Figure 13 - Setting the StartUp project to GUI 15](#_Toc410903455)

[Figure 14 - Adding a new project to the solution 17](#_Toc410903456)

[Figure 15 - VS - Creating a class library 17](#_Toc410903457)

[Figure 16 - Static add function 18](#_Toc410903458)

[Figure 17 - Setting GettingStarted project to be the StartUp 18](#_Toc410903459)

[Figure 18 - Adding a reference 19](#_Toc410903460)

[Figure 19 - MyLibrary reference 19](#_Toc410903461)

[Figure 20 - Using the add function from MyLibrary in the console application 20](#_Toc410903462)

[Figure 21 - Add Fusion API references to MyLibrary 21](#_Toc410903463)

[Figure 22 - Navigate to Fusion API install directory under "Library" 21](#_Toc410903464)

[Figure 23 - References being added to MyLibrary project 22](#_Toc410903465)

[Figure 24 - MyLibrary new assemblies referenced 23](#_Toc410903466)

# Topics covered

1. Detecting the adapter
   * Basic read/write
2. Detecting a PMBusDevice
   * Commands
     + Read vout
     + Read vin
     + Read in
     + Write vout\_command
   * Snapshot command
3. Using a memory map ROM mode
4. Using a memory map Program mode
5. Using a custom memory map
6. Exploring the examples installed by the Fusion API release notes
7. Advanced Topics
   * Multiple adapters
   * Multiple devices
   * Exporting results to an output files
     + CSV
     + Excel spreadsheet
   * Keeping the UI responsive with a background worker thread
   * Try/Catch exception handling

# Glossary

VS- Refers to the full licensed(paid) Visual Studio version 2010

Express – refers to the free C# Express version 2010

# Programs to Install

## C# Editor (Choose One)

### Visual Studio 2010 (Purchase)

### Visual C# 2010 express (Free)

<http://www.visualstudio.com/en-us/downloads#d-2010-express>

### Newer versions

This user guide has a number of screen shots based on the above programs. Newer versions can be used, but the screen shots may differ.

## Fusion API 2.0.7 December 23, 2014

<http://software-dl.ti.com/analog/analog_public_sw/fusion/TI-Fusion-API-2.0.7.exe>

# Part 1 – Getting Started

## Creating a “solution”

Launch Visual Studio editor and click File > New > Project

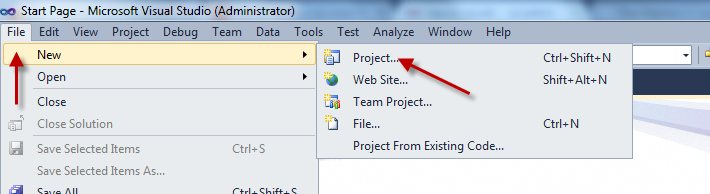


Figure – VS - File>New>Project

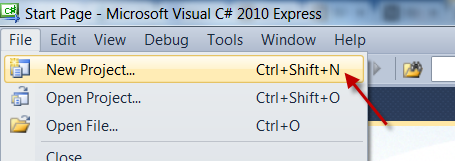


Figure - Express - New Project

Then type in the name “**GettingStarted**” for a new **Console Application**

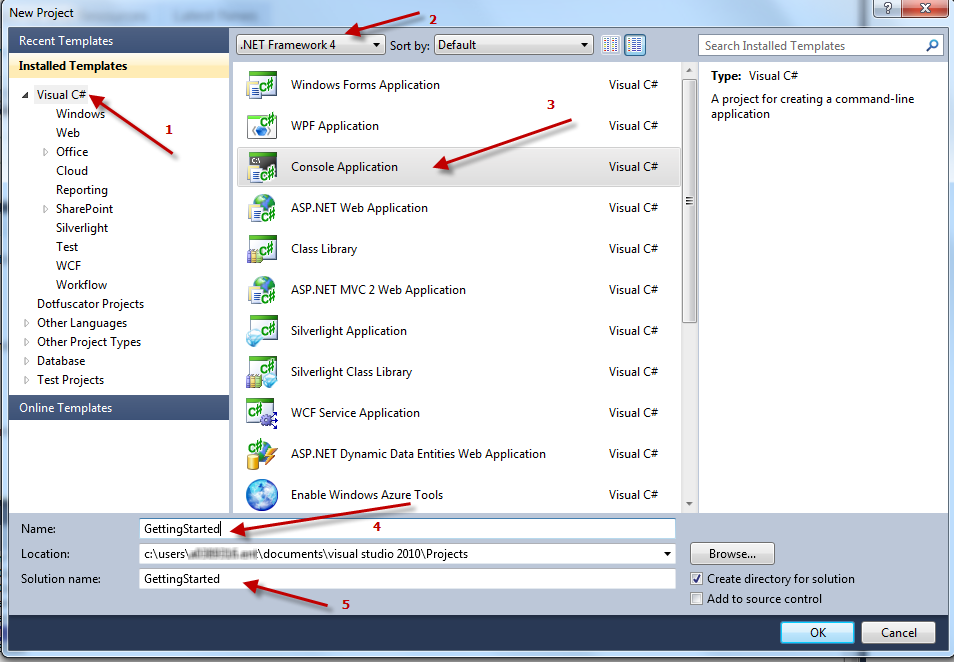


Figure - VS - Console Application settings

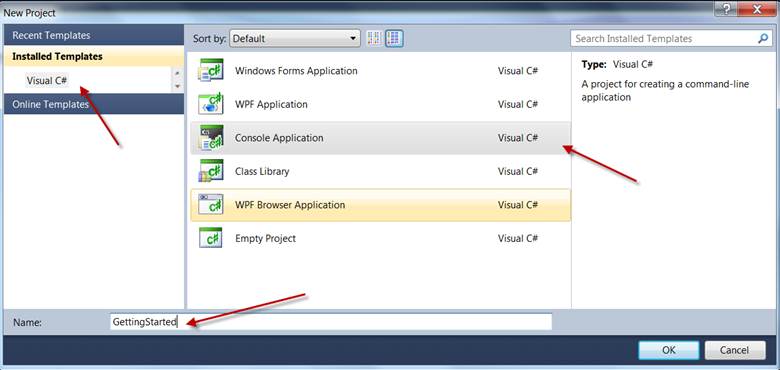


Figure - Express - Console Application settings

After clicking “OK” on the New Project window you can now begin writing your first program in Program.cs!

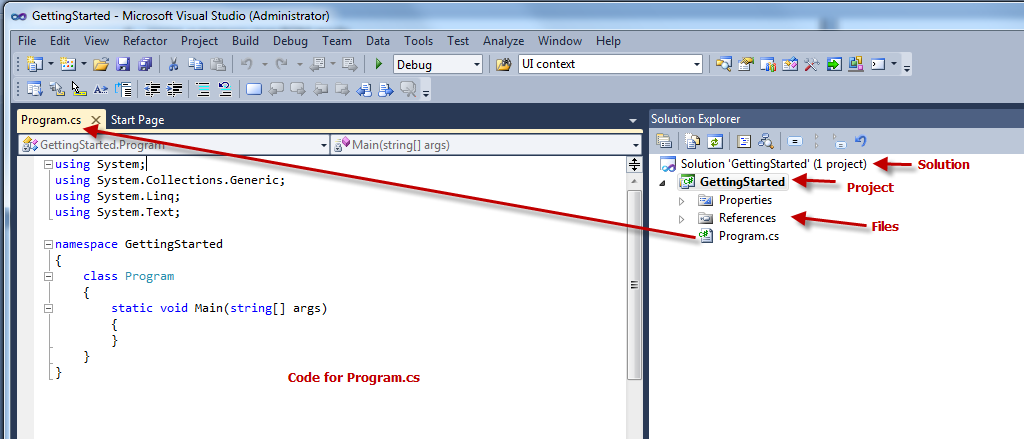


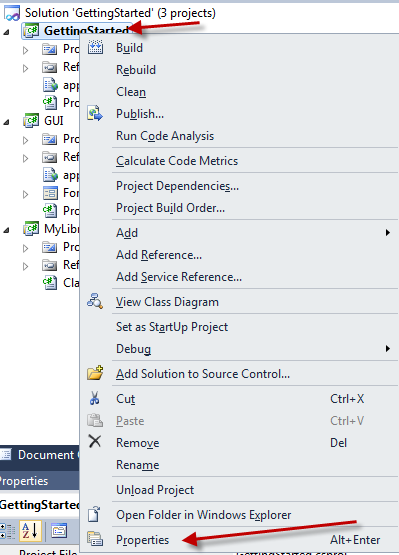
Figure - VS - New Project created

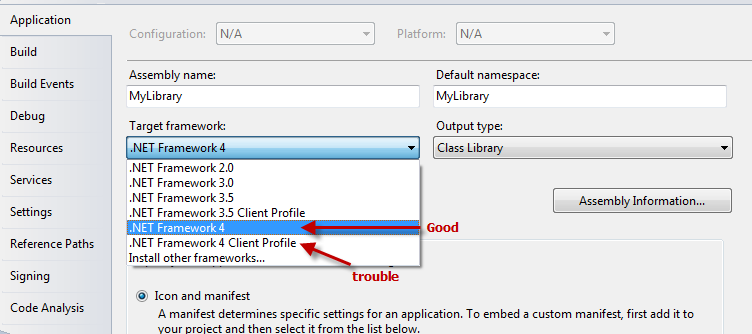
In the editors, coding files are organized in “projects” and “projects” are stored in a solution.

## COMPILE ALL APPLICATIONS FOR .NET 4.0 FRAMEWORK

Before diving in, quickly update your project so that it is compiled for the .NET 4.0 Framework and not .NET 4.0 Client etc.

Click the project “GettingStarted” then right click followed by clicking “Properties.” Under the “Application” tab select Target framework “.NET Framework 4.” See figures below for help.

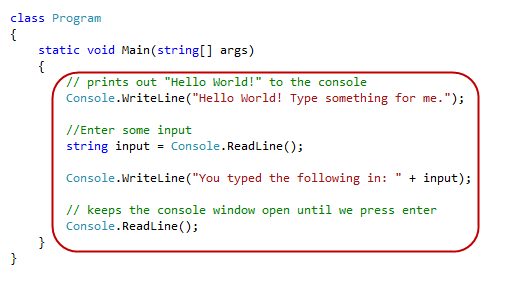




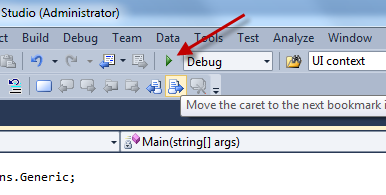
**Technical detail(skip if you like):**By default the editor may choose the “.NET Framework 4 Client Profile.” The client profiles are light-weight framework packages. Problems may arise if you are using a light-weight package and are referencing a library that uses the full .net framework. The errors are usually subtle, so it is best to target the lowest common denominator and stick with the .NET Framework 4.

## Hello World Console Program

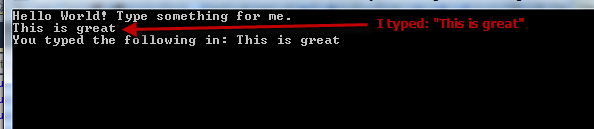
Add the following code to the Main(string[] args) function:



Compile and run your program by first clicking “F6” to build your solution and then clicking the key “F5”, or the “play” button shown in the following picture, to run the program.



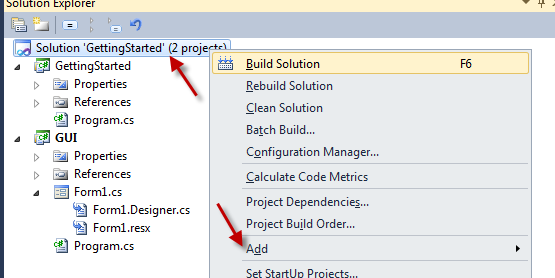
You should see the following screen,



Congratulations on your first program!

## GUI Winforms Application

Right click the “GettingStarted” solution and “Add” a “New Project…”. (Express: you’ll be prompted to save your solution. Please go ahead and do that.) From the New Project window click “Windows Forms Application” and use the name “GUI”.



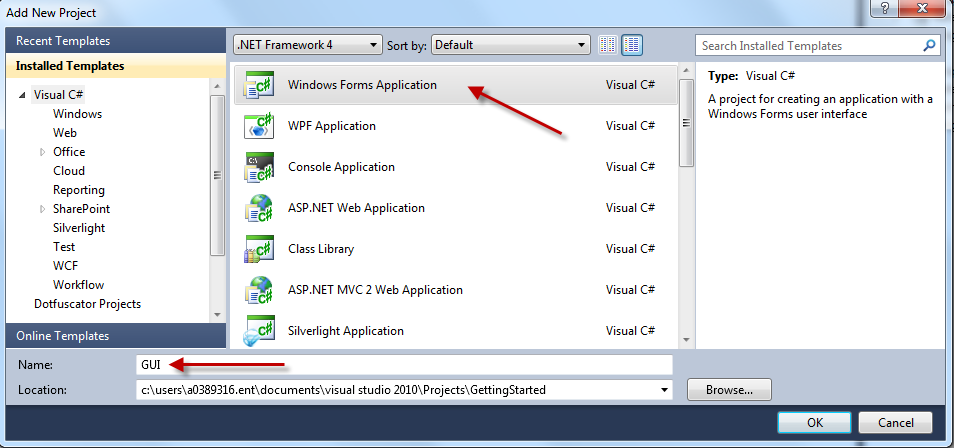


Figure - VS - Add new Windows Forms Application

The following blank form will appear,

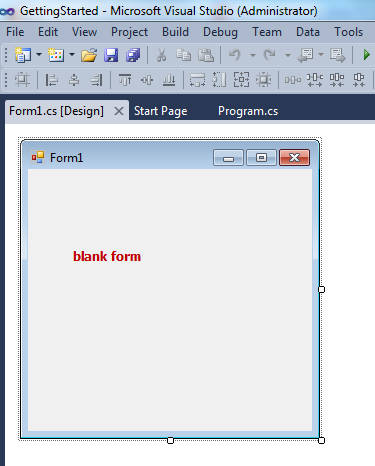


Figure - blank form

Open the toolbox window to grab and drop some controls onto the empty form.

In Visual Studio click View>Toolbox

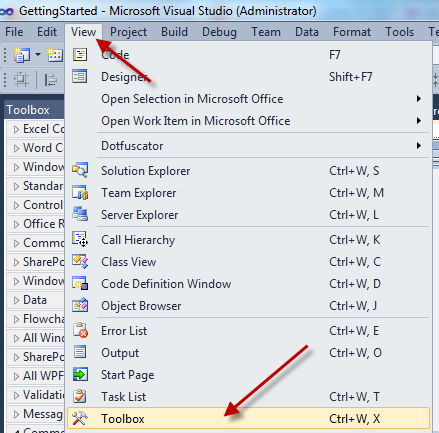


Figure - VS - Toolbox

In Express click View>Other Windows>Toolbox.

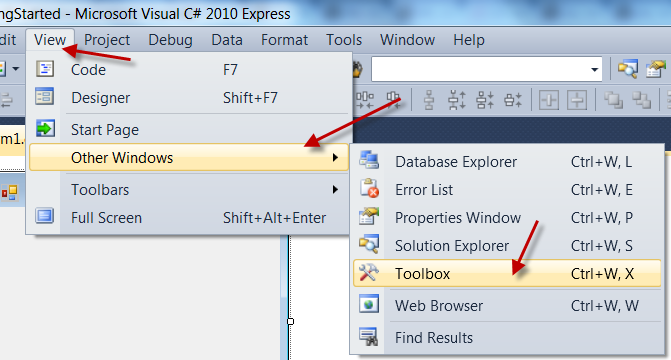


Figure - Express - Toolbox

To keep the “Toolbox” open without the auto hide click the pin so it is oriented as shown in the following picture,

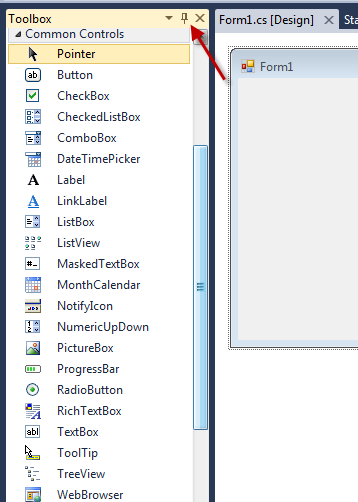
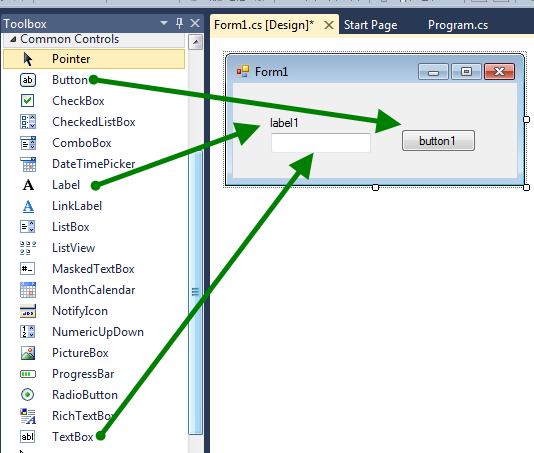


Figure - Pin the toolbox

Drag the following items from the Toolbox on the left, to the empty form on the right.

1. Label
2. Textbox
3. Button



Update the text in the label and the button by editing them in the “Properties Window.” Right click anywhere in the form and click “Properties”

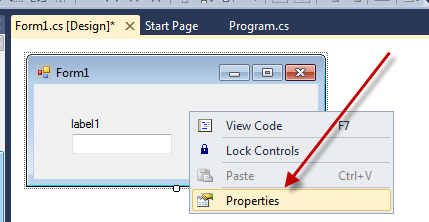


Figure - Accessing the Properties window

The “Properties” window will display the properties of the control last selected. Click the label and button controls and update their “Text” property as follows:

label1 => Enter your name:

button1 => Go!

Change the property for the label1 text can be seen in the following diagram,

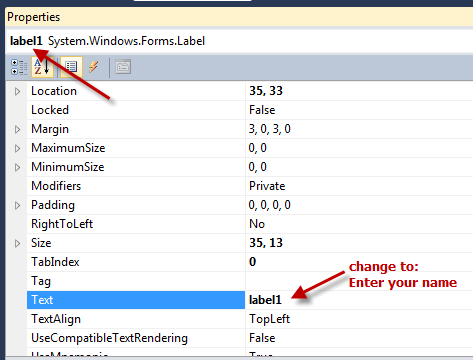
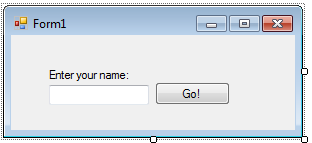


Figure - Text property for label1

Your updated form should look as follows:



Before running this application, you need to set the the winforms application project to be the “startup project.” To do this right click the “GUI” project and click “Set as StartUp Project.” This tells your editor which project to run.

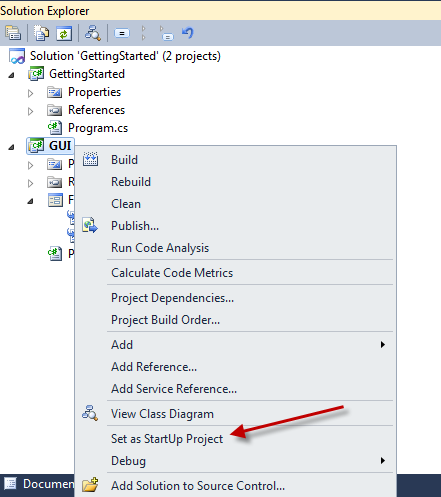
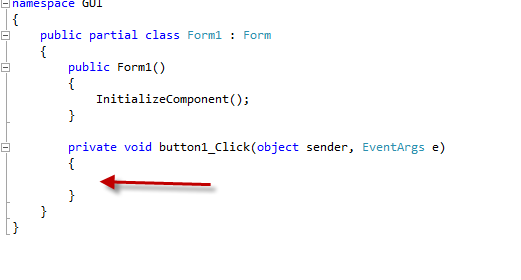


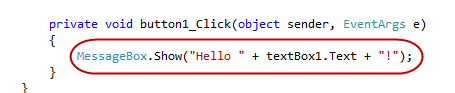
Figure - Setting the StartUp project to GUI

Click “F6” to build and then “F5” to run. You should have a form that looks like above. Let’s add some functionality for the button.

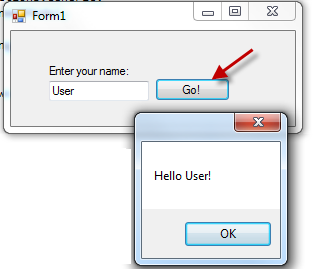
Close your mini form to return back to the editor. Double click the “Go!” button and you should end up in a function as shown in the following picture.



Type in the following code:



Build(F6) and execute(F5) your code and you should have a function button that when clicked will display a pop-up box displaying the following for example,



Congratulations on your Winform GUI application!

## My Library

Let’s create a small library that provides a few simple functions that will be used by your console and winform applications.

Right click the Solution “GettingStarted” node and Add a “New Project…” Click “Class Library.”

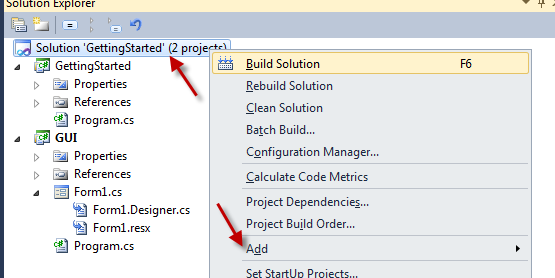


Figure - Adding a new project to the solution

Name your Class Library “MyLibrary”

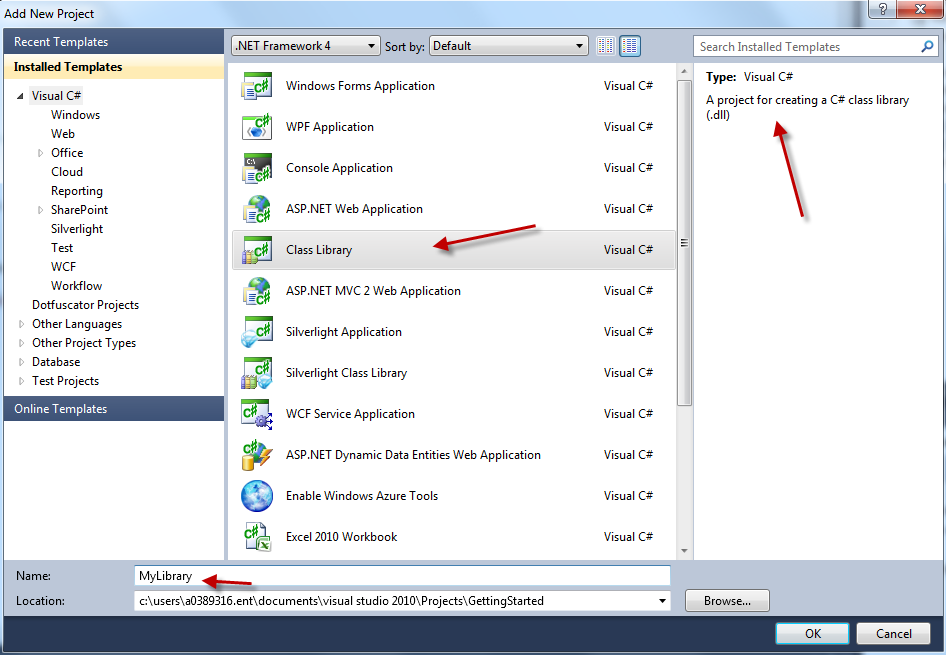


Figure - VS - Creating a class library

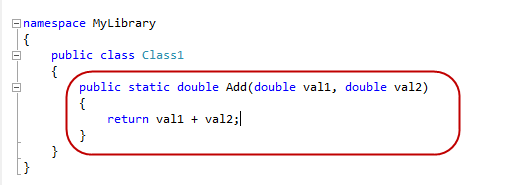
In your new class library add the following static function: 

Figure - Static add function

To test this function you will need the aid of your console application because class libraries cannot be executed.

First set your “GetStarted” console project to be the StartUp project as shown in the following picture,

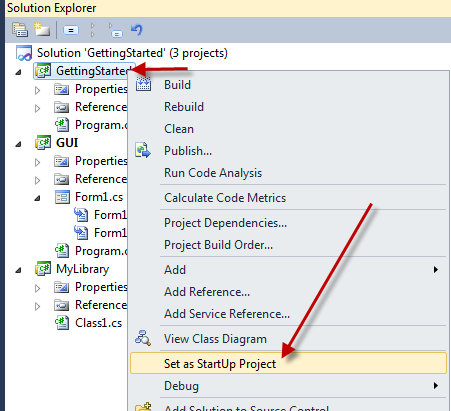


Figure - Setting GettingStarted project to be the StartUp

Although all the projects are adjacent to one another in this solution they don’t actually know about one another. We need to explicitly add a reference to “MyLibrary” in the console application. In the “GettingStarted” project expand the references and Right click “Add Reference…” as shown in the picture below.

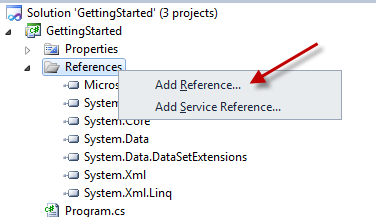


Figure - Adding a reference

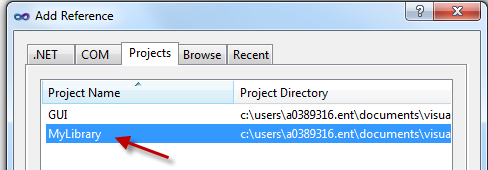
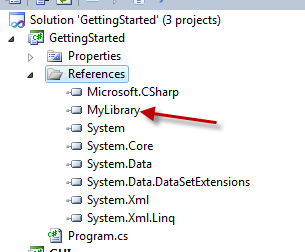


Figure - MyLibrary reference

Click “OK” and the MyLibrary project reference will appear in the Reference list.



Now that the library is recognized by the GettingStarted Console application it can be used.

Double click “Program.cs” and add the following code highlighted in the picture below. Leave the original code and just place the new code above. Also add a “using MyLibrary;” to the top.

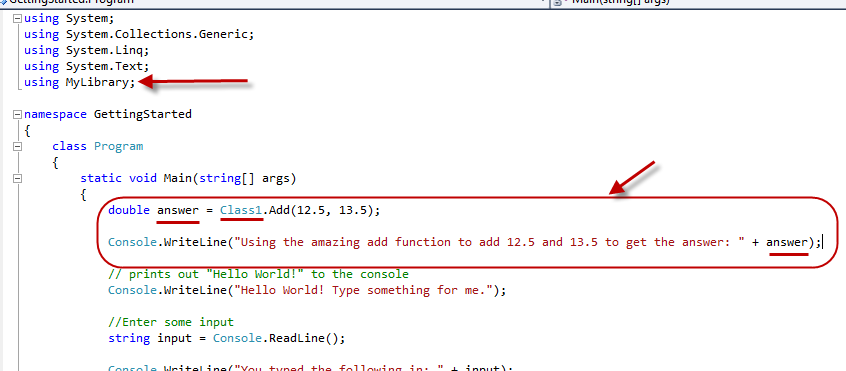
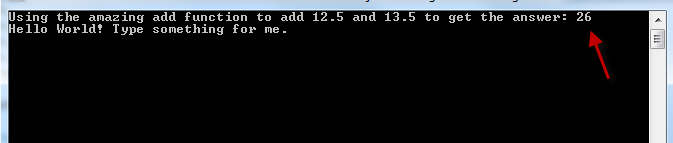


Figure - Using the add function from MyLibrary in the console application

Build (F6) and execute (F5) your application and you should see the following:



Congratulations on referencing a library!

# PART 2 – Fusion api

## Adding Fusion API library references

Since you are familiar with adding references, let’s add a reference to some of the libraries in the Fusion API.

Add the following references to your “MyLibrary” project. You will need to navigate to the “Library” folder in your Fusion API install directory. See the following pictures for help.

1. Com.Muegel.Misc
2. TIDP.Factory.Core
3. TIDP.Factory.CustomerTypes
4. TIDP.Factory.TaskLibrary
5. TIDP.Misc
6. TIDP.PMBus
7. TIDP.SAA

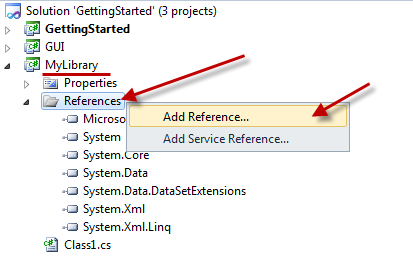


Figure - Add Fusion API references to MyLibrary

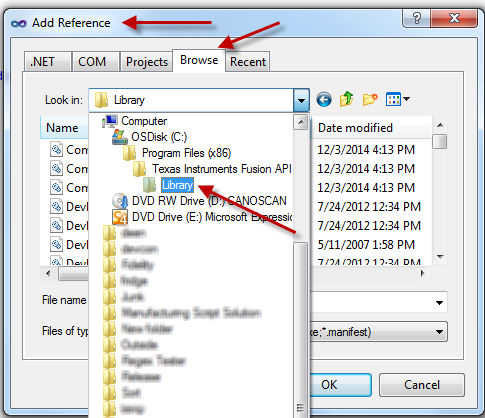


Figure - Navigate to Fusion API install directory under "Library"

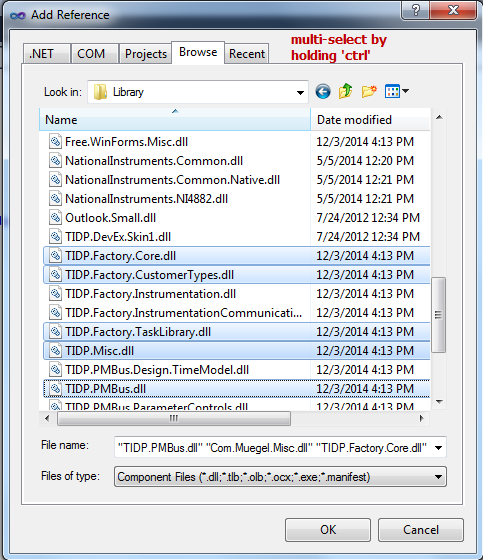


Figure - References being added to MyLibrary project (Note:not all are shown in this image)

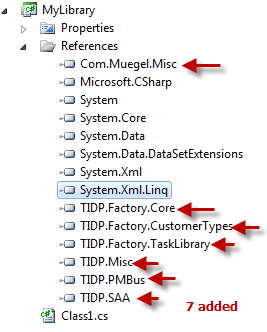
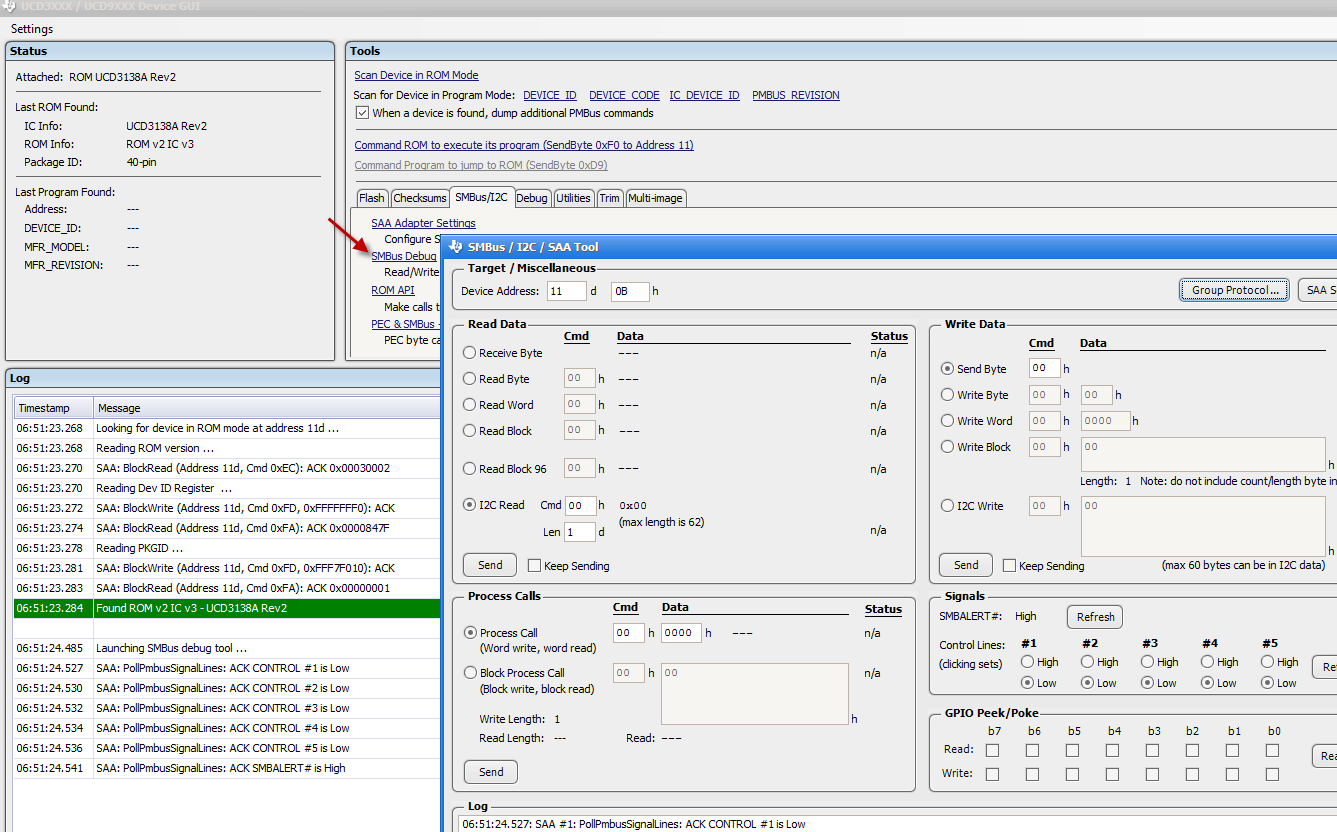


Figure - MyLibrary new assemblies referenced

## smbus adapter

You are likely familiar with the SMBus Debugger from the UCD3xxx Device GUI. This tool can be used to do all your reads and writes to the device. It is quite low-level.

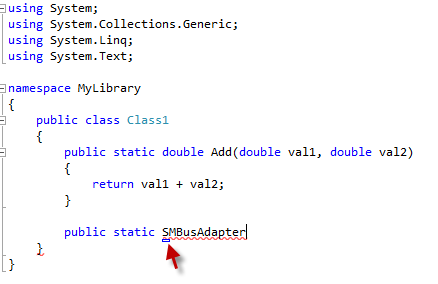


Let’s programmatically use the SMBus Adapter in your C#. Navigate to your Class1.cs and create a function called SMBus\_Adapter\_Scan as shown in the following figures. The signature will be as follows:

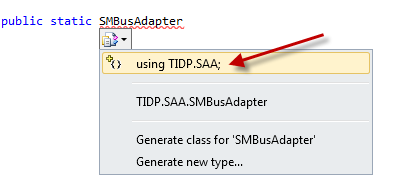
public static void SMBus\_Adapter\_Scan()

### quick trick for “usings” in c#

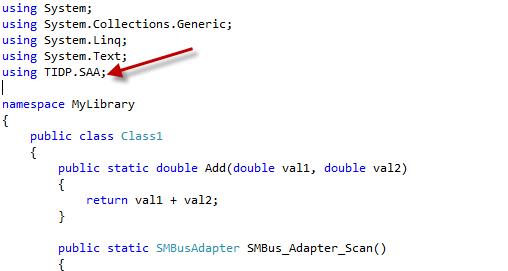
While typing the function signature out you’ll notice a little blue line that appears at the beginning of the return type SMBusAdapter as shown below,



Before typing out the signature heading a using statement of TIDP.SAA should’ve been placed at the top. As a quick trick you can simply mouse over the little blue line and a drop down menu will appear giving you that option to do just that!



Go ahead and click “using TIDP.SAA.” Notice that this line will appear in the top of your code.



Let’s add the following code: (Which you can copy and paste ☺)

//call Discover function and will populate its SMBusAdapter.Adapters property

//with all the available SAA Adapters.

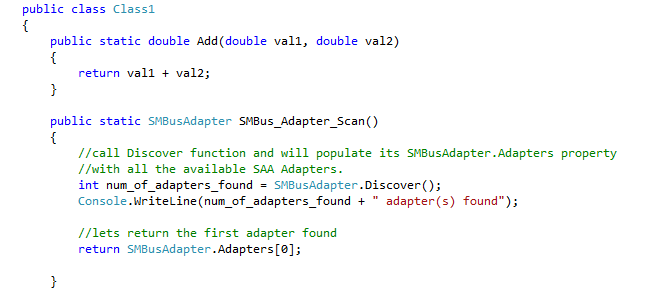
int num\_of\_adapters\_found = SMBusAdapter.Discover();

Console.WriteLine(num\_of\_adapters\_found +" adapter(s) found");

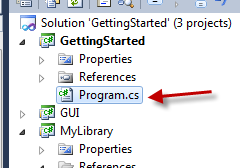
//lets return the first adapter found

return SMBusAdapter.Adapters[0];

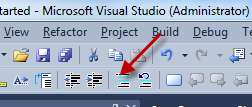
The finished function should look as follows:



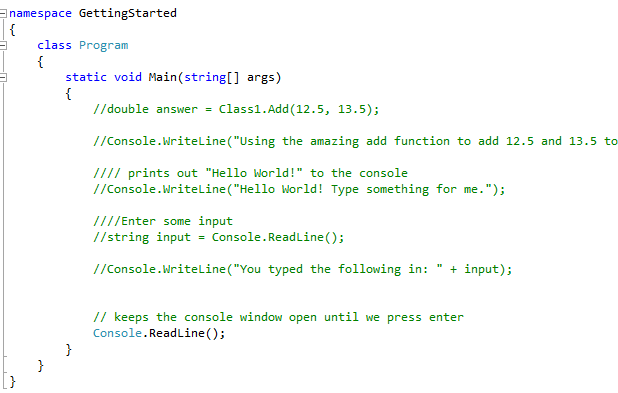
This is a handy function that your applications can use. Use it in your console application “GettingStarted.” Open your Program.cs and comment out all the lines except for the last Console.Readline().



A quick way to do this is to select all the lines and press “CTRL E C” or click the comment button at the top shown here,



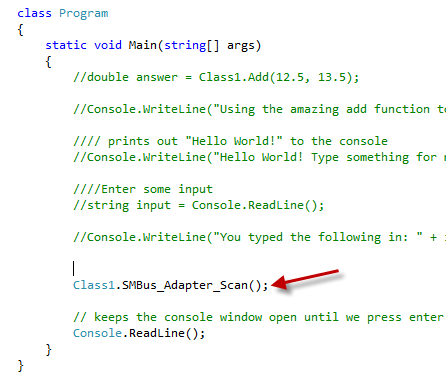
Your code should look as follows:



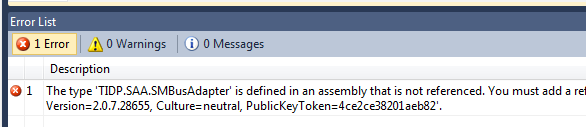
Add the following line:

Class1.SMBus\_Adapter\_Scan();

Your program should look as follows.

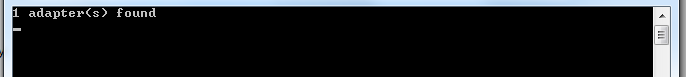


Compile(F6) and execute(F5). You should get an error in the “Error List” as shown below.



Since you are calling a function using the SMBusAdapter, you will also need to reference that library in the console application. Update the references for your console application as described in the section Adding Fusion API library references.

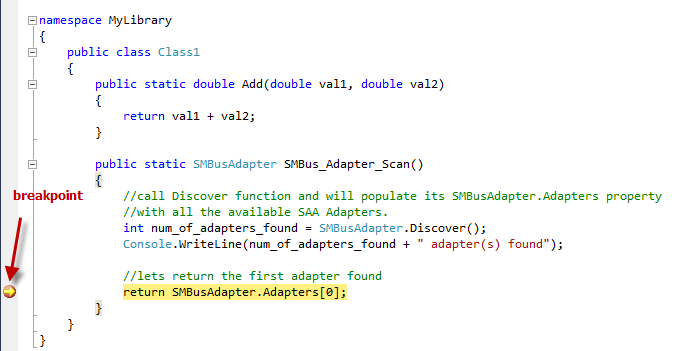
Now you should see the following screen, if you have 1 adapter connected.



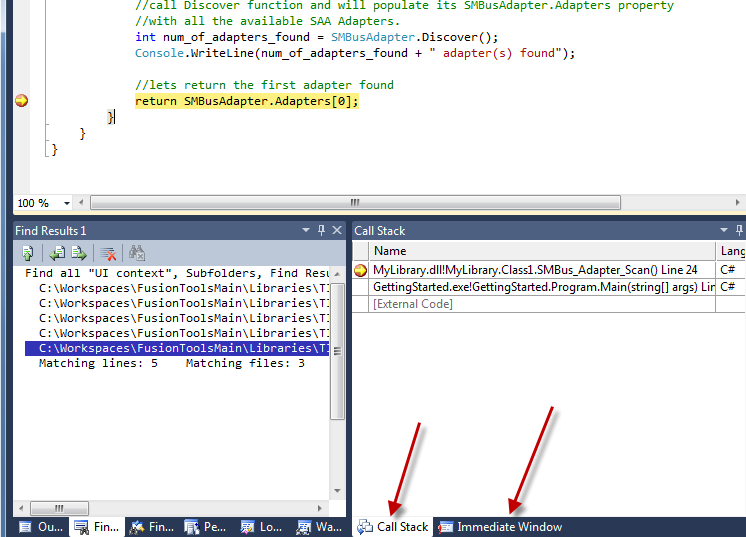
That may not be too exciting but there is a lot you can do with the SMBusAdapter. This is a good opportunity to display the power of the breakpoint and the “Immediate Window.”

### Breakpoints, call stack, and the immediate window

Go to your function implementation for the SMBus\_Adapter\_Scan and set a breakpoint on the line that has “return SMBusAdapter.Adapters[0]”. Press F5 to execute your code and should see the following where the execution has hit your breakpoint.



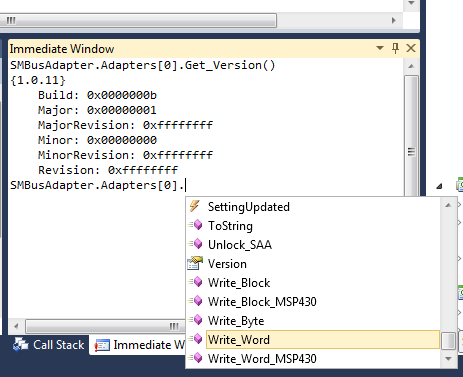
The “Call Stack” and “Immediate Window” tabs should be visible to the bottom right as shown here:



You can navigate the call stack to see how the execution got to this location. By double-clicking a line you’ll go to that line number in your code. Wherever you double-click you’ll have access to the variables that are alive at that location. This is very useful in debugging what values are being swapped around in your code without having to do a whole series of Console.Writeln(“I’m here and the value is…”);

To access the variables in the current scope of your breakpoint, flip over to the “Immediate Window.” This is basically an interpreter, like what is available in MatLab. You can only use this when you’ve hit a breakpoint.

As can be seen in the following figure you can interact with the SMBusAdapter “immediately.”



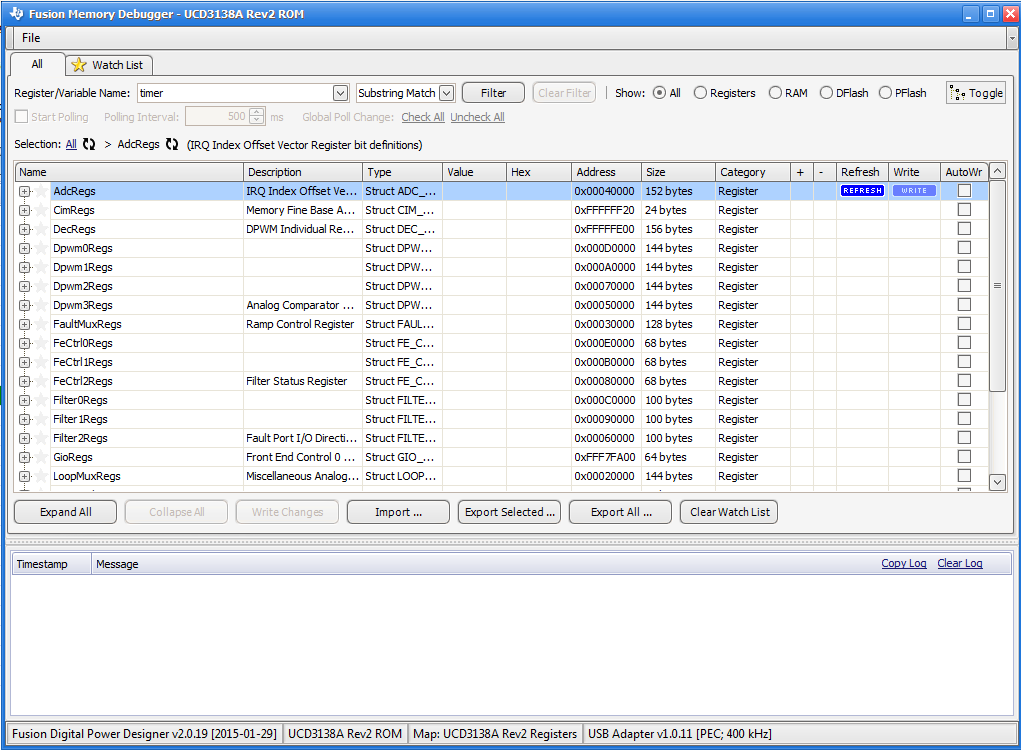
Sometimes it is useful to code in here and then copy and paste over into your coding files.

**Class:**Time for a SMBusAdapter demonstration with the SendByte, look up at the board if you are in class, otherwise continue to next sections.

Congratulations on using the SMBusAdapter!

## Memory debugger – Memory maps!

If you are familiar with the SMBus Debugger you are probably quite familiar with the Memory Debugger.



Add the following lines of code to a function called Memory\_Debugger\_Test in your “MyLibrary” project right after the function SMBus\_Adapter\_Scan() (You can copy and paste ☺).

public static void Memory\_Debugger\_Test()

{

UCD3138ARegistersEasyMemoryMap map = new UCD3138ARegistersEasyMemoryMap();

var config = map.Variables.AdcRegs.ADCAVGCTRL.bit.AVG0\_CONFIG.Value;

if (config == 0)

{

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*This is not good.");

}

map.Variables.AdcRegs.ADCAVGCTRL.bit.AVG0\_CONFIG.Value = 3;

map.Write\_Pending();

//or you can write directly

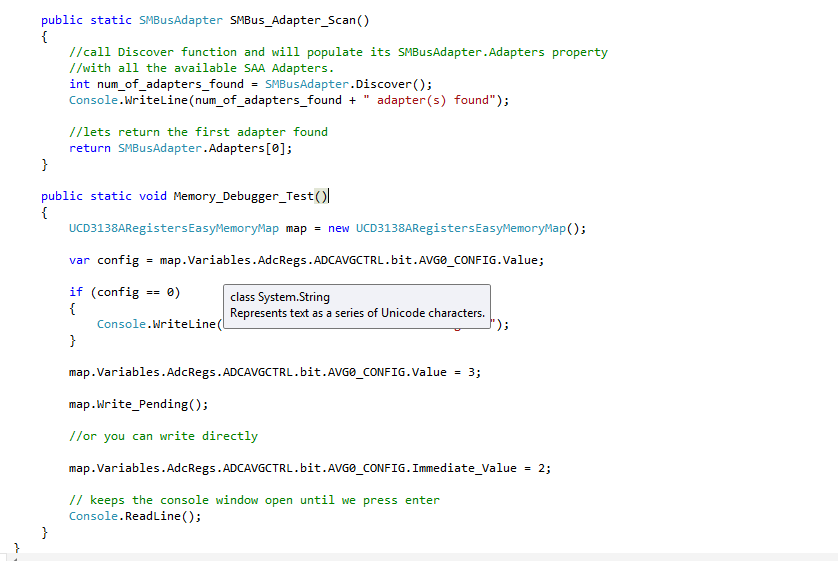
map.Variables.AdcRegs.ADCAVGCTRL.bit.AVG0\_CONFIG.Immediate\_Value = 2;

// keeps the console window open until we press enter

Console.ReadLine();

}

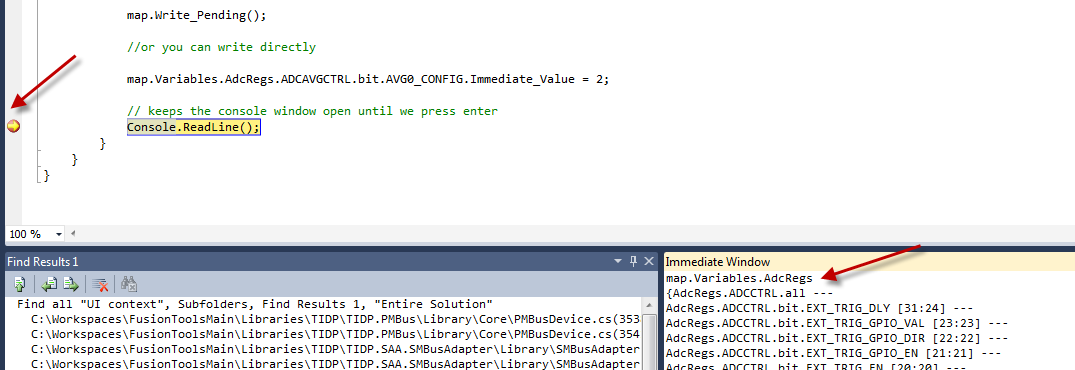
Your program should look as follows:



Add a call to this function from your Program.cs in your console application “GettingStarted”.



Set a breakpoint in the Memory\_Debugger\_Test function on the last line and interact with it in the “Immediate Window.”



## PMBus commands

Add another function to Class1.cs in the project “MyLibrary” as follows(you can copy and paste):

public static void PMBus\_Commands()

{

//sends the device to program mode

PMBusDevice.Discover();

PMBusDevice device = PMBusDevice.Devices[0];

double vout = device.Commands.READ\_VOUT().Latest.Value;

double iout = device.Commands.READ\_IOUT().Latest.Value;

double power = vout\*iout;

Console.WriteLine("Vout = "+vout.ToString("n3") +" V");

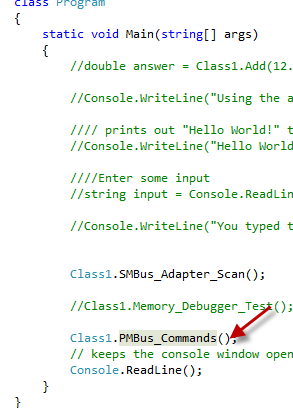
Console.WriteLine("Iout = "+iout.ToString("n3") +" A");

Console.WriteLine("Power = "+power.ToString("n3")+" W");

}

Call this function from Program.cs in your console program “GettingStarted” by adding the line

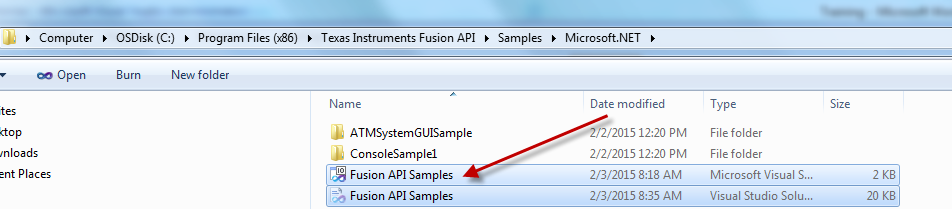
Class1.PMBus\_Commands();



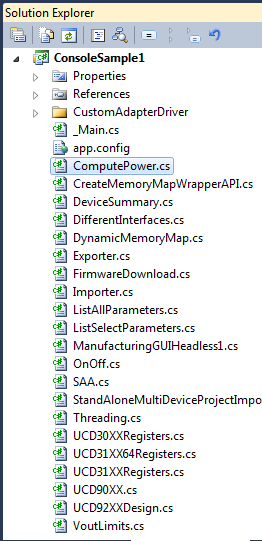
Set a breakpoint in the PMBus\_Commands function at the last line and explore the PMBusDevice with the “Immediate Window”.

## more examples - fusion api samples solution

There are a number of examples that are part of the Fusion API. If you navigate to your install folder you will find



In the Fusion API Samples solution there is a console project called “ConsoleSample1”



# Advanced topics

### How to generate your own memory map to be used in the api

See example: CreateMemoryMapWrapperAPI.cs in Fusion API Samples.

### Connecting to multiple devices

### connecting to multiple SAA adapters