# 01 Working with SIMPL# Series

This document assumes the user is familiar with SIMPL Windows and SIMPL+ programming.

VisualStudio 2008 SP1 is required with Crestron SIMPLSharp plugin.

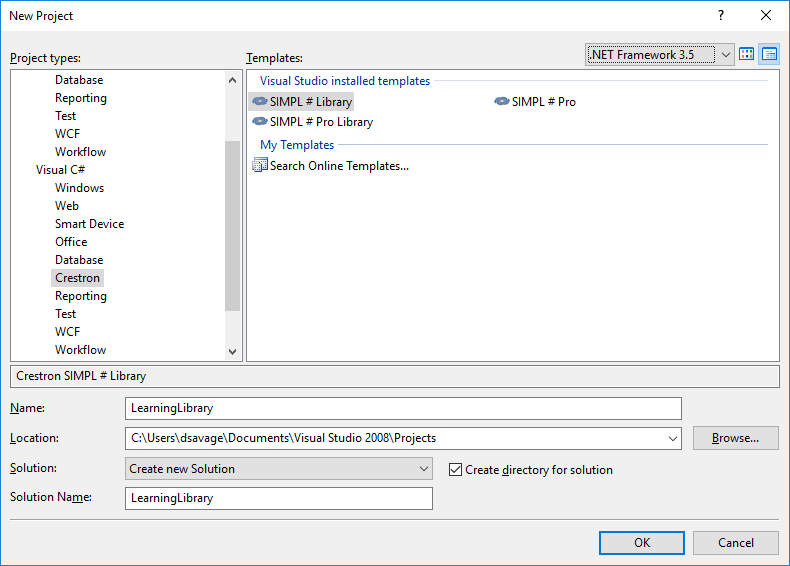
## Using a SIMPL# Library

### Layered concept of accessing the SIMPL# library

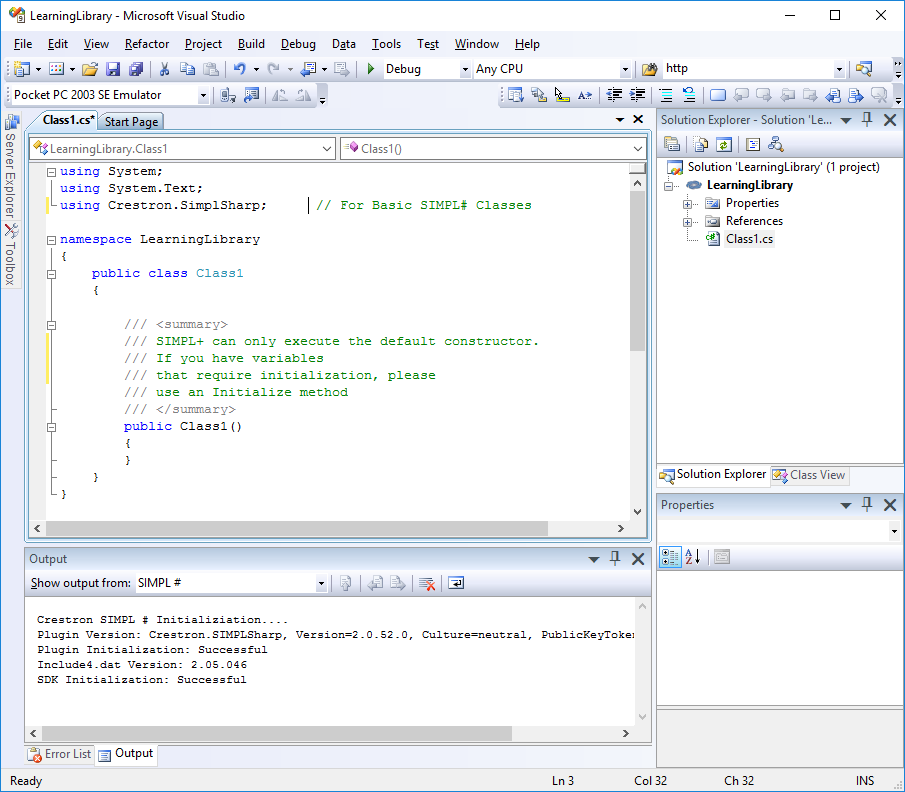
1. SIMPL code will contain a SIMPL+ module.
2. SIMPL+ module will contain a reference to a SIMPL# library file
   1. SIMPL# library file ends in .clz.
   2. Have the .clz file in the same folder as the SIMPL+ module and reference the library with #USER\_SIMPLSHARP\_LIBRARY “<name without extension>”
   3. If the .clz is in a different directory you need to add a reference to its folder location using #INCLUDEPATH “<absolute file path>”

### Creating the library file

Once all required software is loaded create a new project in VisualStudio 2008.



The IDE will present the following;



**‘namespace’** – logical grouping (container) for all of the functions (called methods) you would like to perform in this library. There are other namespaces included in SIMPL# that you can reference and pull their functions from. Note the hierarchy in the Solution Explorer.

**‘using’** – this creates a reference to other namespaces so we can use their methods in your SIMPL# code.

**‘class’** – this is another logical container (within your namespace) to divide your methods into groups within SIMPL#. To access methods within a SIMPL# class from SIMPL+ you need to point to this class (create/instantiate an object of this class) from your SIMPL+ code (to be shown later).

**Class1.cs** – this is your SIMPL# source code filename. This can be changed.

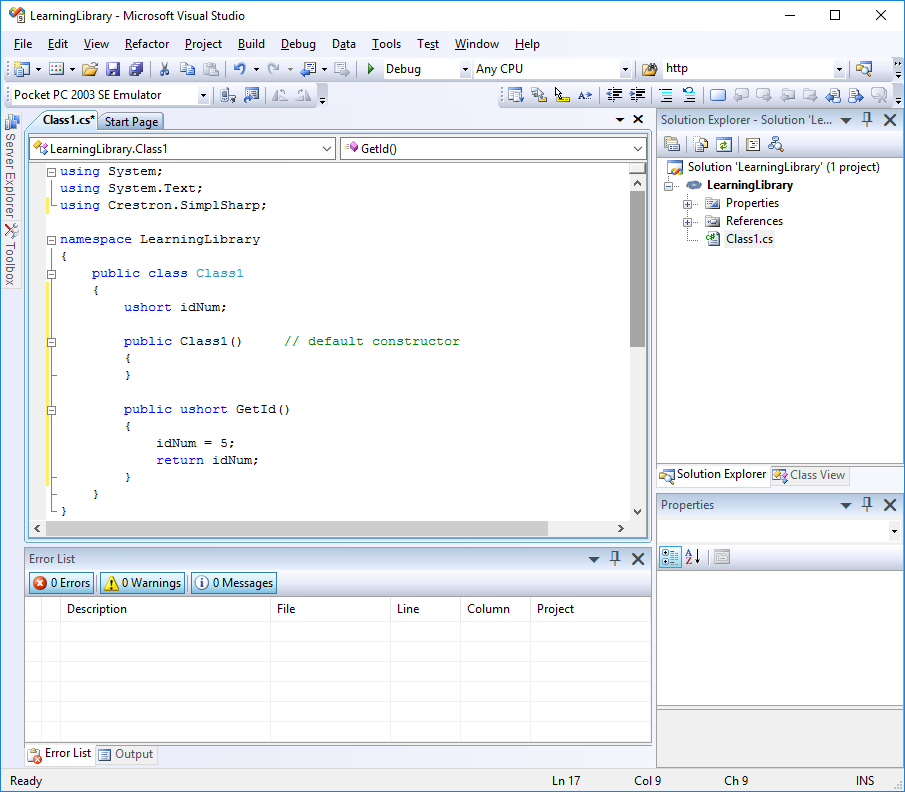
**‘public’** – this deals with the topic of scope (who can access what). Public means that SIMPL+ will be able to see this and interact with it directly from SIMPL+.

**‘public Class1()’** – this is the default constructor for a class (required to be in SIMPL# by SIMPL+). Note its name and location in relation to the ‘**public class Class1**’ declaration – it is nested inside. You don’t need to do anything with this other than have it in your SIMPL# code.

NB. METHODS = C# term for functions. FUNCTIONS = SIMPL+ term for methods. Same same.

### TASK: Programming a button press in SIMPL to get a value or two from SIMPL#

1. **Create the logic in SIMPL#**



**‘idNum’** – this is my variable (bucket) I will use to store a value. This is created/declared in the class, but outside of any individual method within the class. This means that any function within the class can read and write to this variable.

**‘ushort’** – data type for the variable I need to store. This type is equal to an unsigned (regular plain ’old) integer in SIMPL+.

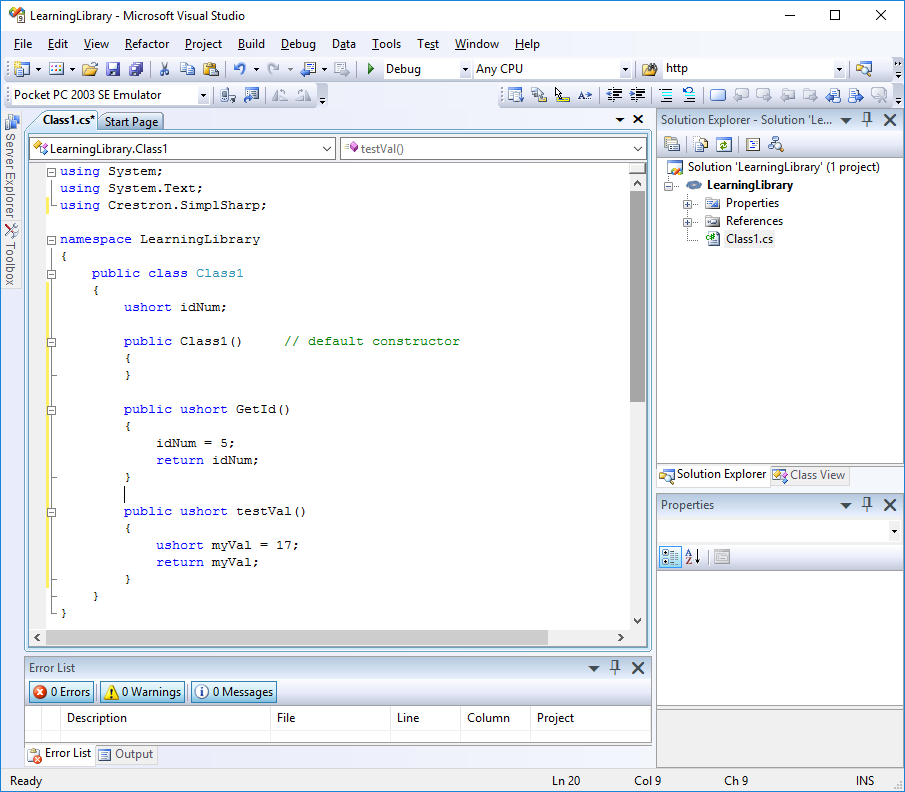
**‘public ushort GetID()’** – this is the method that I want to run in SIMPL#. It is public so SIMPL+ can get to it. It also has a data type (ushort) so that once the SIMPL# part is done it will give a value back (return a value) to SIMPL+. There are many data types including one called ‘void’. Void does not give anything back and is used to complete tasks or calculations that stay within SIMPL# itself.

**‘idNum = 5;’** – I declared my variable (with ‘**ushort idNum;**’) and now I’m assigning it the value of 5.

**‘return idNum;’** – when your method has a data type that is not the void type, it needs to give back some information of the same data type to whatever ran/called the method. That is done using this expression. If the method was called from SIMPL+ then;

* + ‘public ushort GetID()’ will give SIMPL+ back an unsigned integer.
  + ‘public short GetID()’ would give SIMPL+ back a signed integer.
  + ‘public string GetID()’ would give SIMPL+ back a string.

……adding in more logic…



There is now a 2nd method in our class that is also public so that SIMPL+ can get to it.

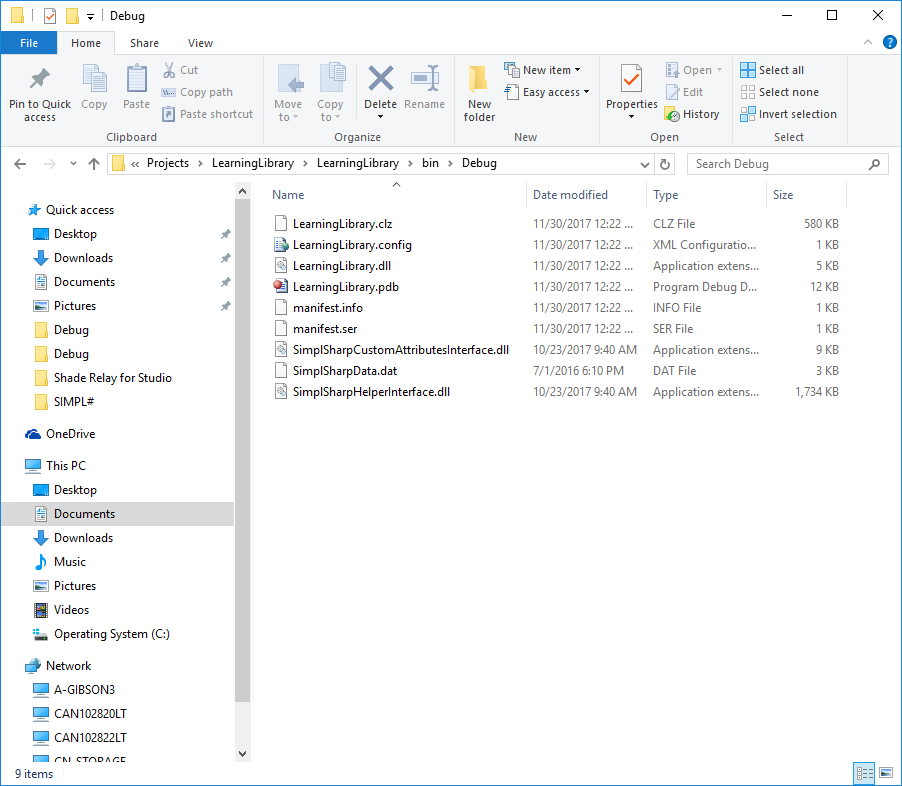
**‘ushort myVal = 17;’** – in SIMPL+ you need to declare your variables (with their data types) first, THEN assign a value to them. In SIMPL# you can either declare first then assign, OR do it all in one step like this! Since this variable was created/declared within the testVal() method it is called a local variable (it’s scope is local) and it can only be read from/written to from inside this method.

**‘return myVal;’** – local scope on a variable does not mean that SIMPL+ can’t be given this value. Return will pass the value back to whomever ran the method (elsewhere in the SIMPL# code or from SIMPL+).

1. **Build the SIMPL# library**

Use the menu Build/Build Solution or Shift+Ctrl+B. It will generate the following files in your SIMPL# project folder;

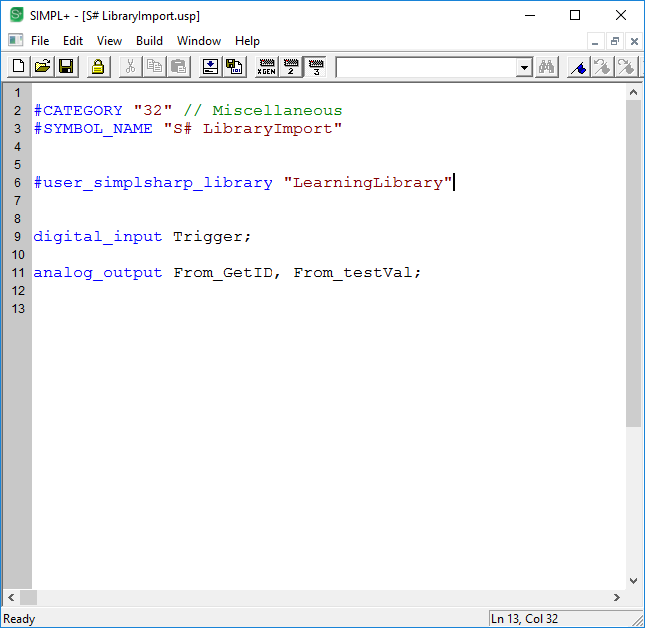
*(my output location is …/Visual Studio 2008/Projects/LearningLibrary/LearningLibrary/bin/Debug)*



1. **Create the base logic in SIMPL+ and point to the SIMPL# library**

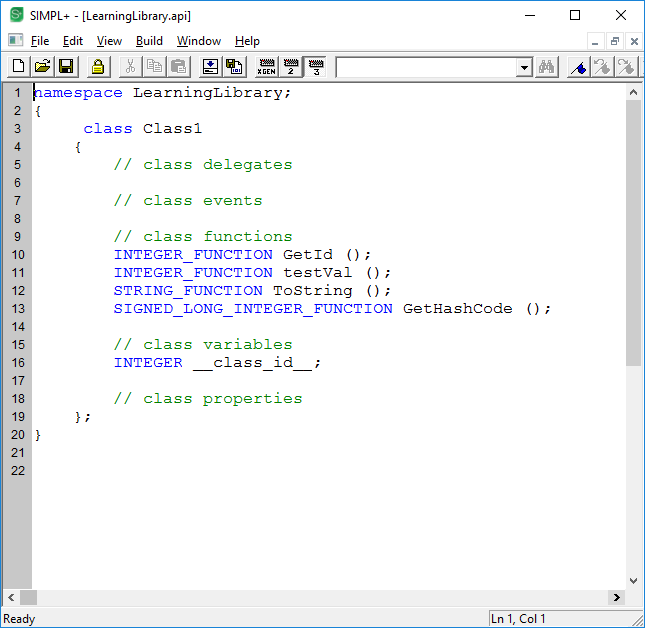
I opened the SIMPL+ editor, deleted the template text and used the Edit menu to help insert the ‘#CATEGORY’ line (this is the folder where your SIMPL+ module will show up in your database).

**Give your symbol a name, define the inputs and outputs and save it in your output location from building the SIMPL# project.**



Adding in the reference to the SIMPL# library will now allow you to access the API (Application Programming Interface) for the SIMPL# code you wrote. Basically, it will show you what’s available to you to use.

1. **Access the API for the SIMPL# code by right-clicking in your SIMPL+ editor and choose “Open API for LearningLibrary…”**



**‘STRING\_FUNCTION ToString’** – included automatically by SIMPL#.

**‘SIGNED\_LONG\_INTEGER\_FUNCTION GetHashCode();’** – included automatically by SIMPL#.

**‘INTEGER \_\_class\_id\_\_;’** – included automatically by SIMPL#.

**‘INTEGER\_FUNCTION GetId();’** – This is my 1st SIMPL# method that I would like to run from SIMPL+.

**‘INTEGER\_FUNCTION testVal();’** – This is the 2nd SIMPL# method I want to run from SIMPL+.

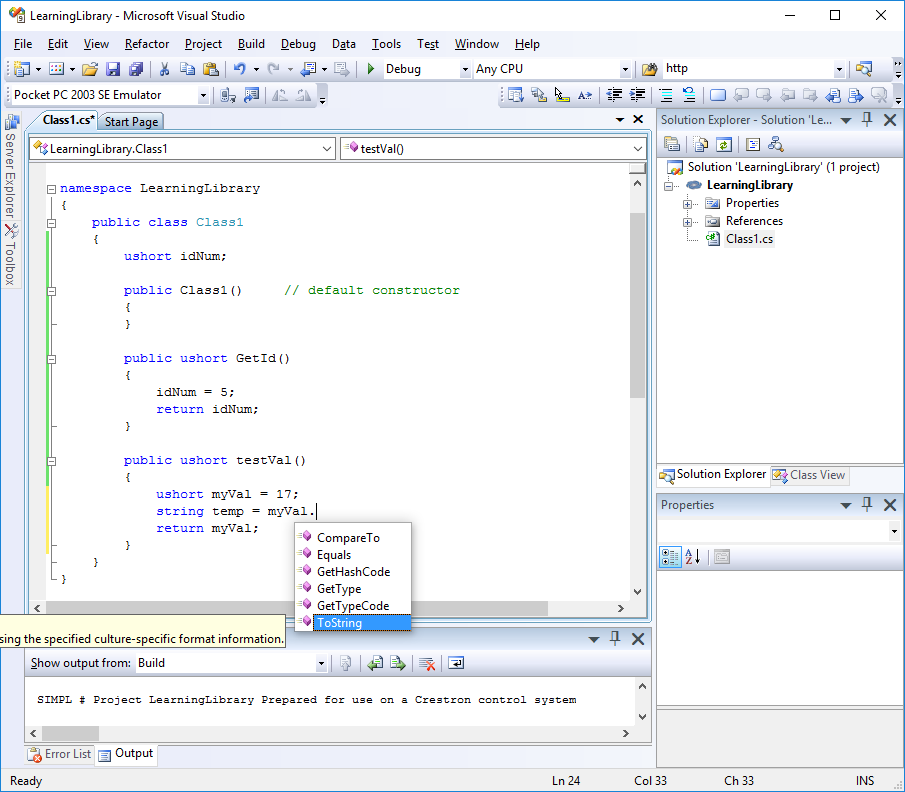
These last 2 methods showed up because they were declared as public in SIMPL#. Once you see what is available to use you can close the API window.

1. **Create an Object of the SIMPL# Class in SIMPL+**

In order to access these SIMPL# functions inside the SIMPL# class you need to do what is called ‘instantiate an object of the class’. This works similarly to how a variable works; if I say ‘integer x;’ then the variable ‘x’ contains values of the data type ‘integer’. It becomes an ‘object of the type integer’.

The most confusing part about this is understanding what makes up the ‘data type’ of Class1. We understand the ‘integer’ type as a whole number, but the data type of Class1 is not that simple to understand.

Well, in C# the data type of integer actually contains some methods as well…

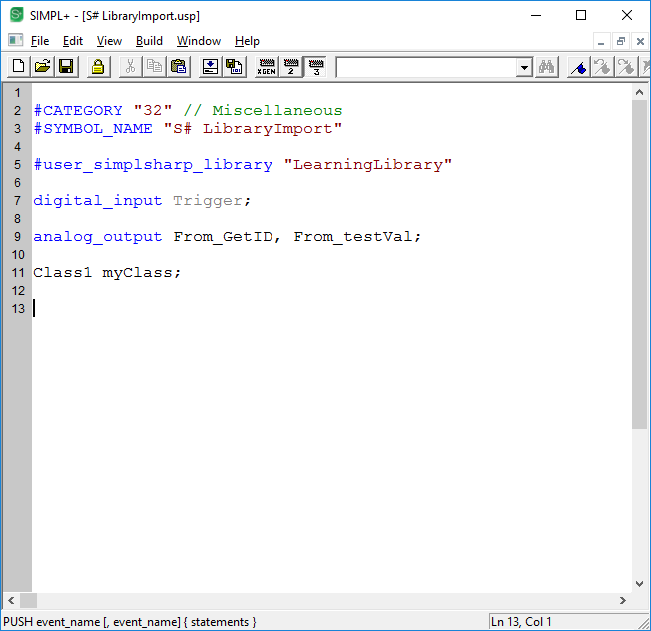


I can access these methods using the dot ‘.’ notation after the variable name. ‘**myVal.**’ unlocks the Intellisense feature in VisualStudio that provides a quick menu of available next steps for that data type. As you can see, integer is not as simple as you though it was as far as C# is concerned! C# allows this so you can create your own data types if you need to or access more complex features of data types already in the software.

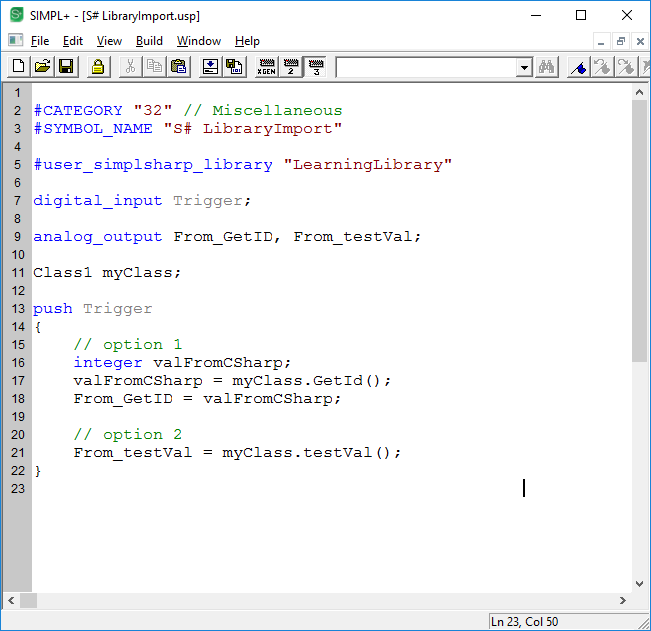
**‘string temp = myVal.ToString();’** – this line takes our integer value inside the variable myVal and turns it into a string datatype, then assigns it to the string variable of temp. (The line ‘**string temp = myVal.ToString();**’ is just a talking point. Don’t add this!).

*\*\*\*Under the hood – Remember the discussion earlier about methods that ‘return’ data? The ToString(); method is a string method. It RETURNS a string (back to whomever called the method) after changing the integer value inside myVal to a string. In this case, assigning a value to the variable ‘temp’ was what called the method and ‘temp’ gets the returned string.*

In the SIMPL+ screenshot below, an ‘object of Class1’ is a SIMPL+ variable (I named it ‘myClass’) that is of the data type ‘Class1’.



1. **Create a SIMPL+ event and call SIMPL# methods through the class object (object of the class called Class1)**

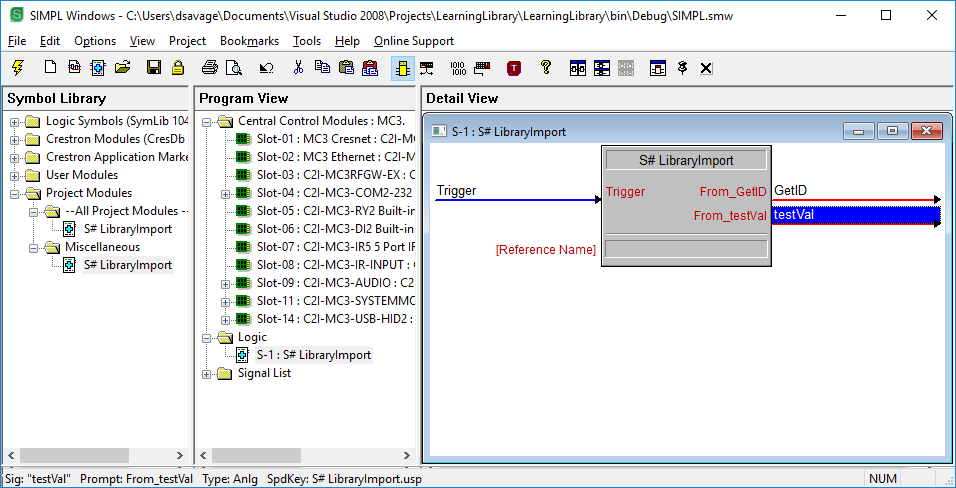


‘**valFromCSharp = myClass.GetId();**’ – the same ‘dot’ notation used in SIMPL# is now available in SIMPL+. This is how SIMPL+ reaches into the SIMPL# Class to get to its methods.

1. **Ensure that only 3-series is selected at the top of the compiler and compile the SIMPL+ code using F12 or menu Build\Save and Compile.**

If all goes well, there should be no errors. SIMPL# is case sensitive so be aware of how you type the names into SIMPL+. GetId is okay, GetID is NOT!

1. **Create a SIMPL program with the 3-series processor of your choice, save it in your Debug folder with your SIMPL# and SIMPL+ code, bring in your Project level SIMPL+ module and define signals on the input and outputs.**



1. **Compile/load/debug (in debugger)**

