# 02 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.

## SIMPL# Library Delegates and Events

Continuing from the code written in document 01 SIMPL Sharp Libraries – Introduction we will be extending in this document to include Delegates and Events.

In the introduction we took a button press from SIMPL Windows into a SIMPL+ module and used the SIMPL+ module to call methods in a SIMPL# library. The methods called were designed to simply return values back to whomever called them (ie. the code in SIMPL+). SIMPL+ treated these returned values in 2 different ways;

Option 1 – use a variable in SIMPL+ to store the returned value from a SIMPL# function and then send the value to an analog output on the SIMPL+ module.

Option 2 – assign an analog output on the SIMPL+ module to the returned value from a SIMPL# function directly.

Either way will work just fine.

## So, what are Delegates used for?

So far, SIMPL+ has called methods from inside your SIMPL# library. Since SIMPL# libraries cannot access inputs and outputs on the SIMPL+ module (or in SIMPL) directly we need SIMPL+ to complete any input/output signal related functions. SIMPL+ itself is capable of performing its own functions and can be made to work together with SIMPL# to complete the tasks required. Generally, SIMPL+ will handle passing the input/output signals between SIMPL and SIMPL# with human interaction happening in SIMPL and the harder calculations/parsing/secure connection details happening in SIMPL#.

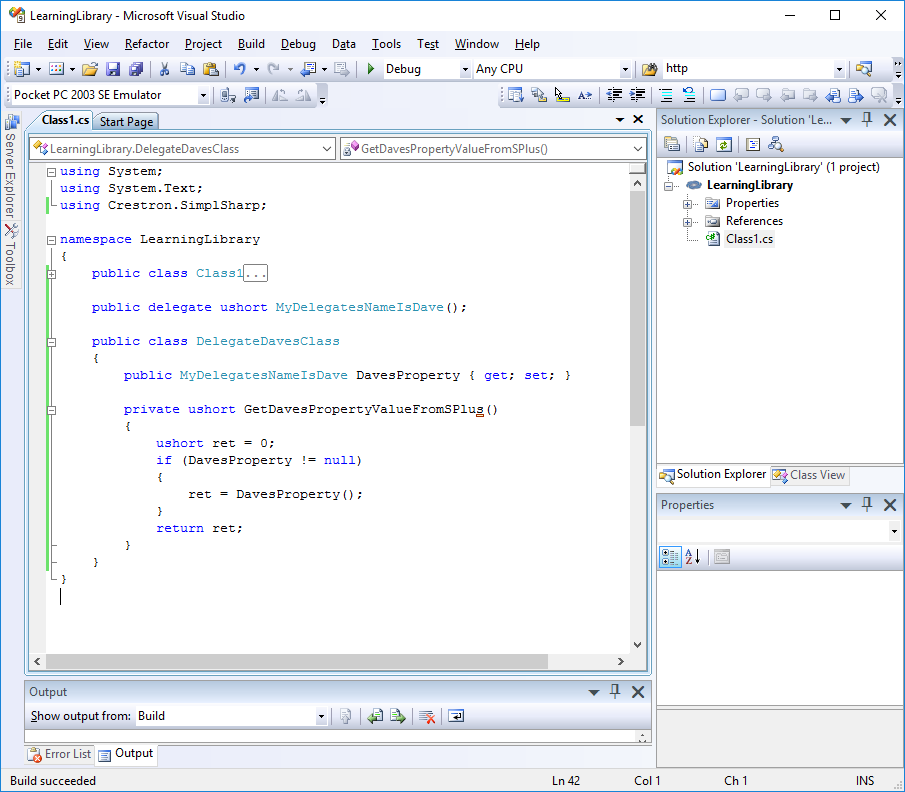
Delegates add a special functionality to the SIMPL# code. It allows SIMPL+ functions to be accessed from SIMPL# methods (the opposite direction of what was completed in the Introduction document). SIMPL# methods will be able to call-back to SIMPL+ functions as well as have SIMPL+ functions return data to SIMPL# methods.

## Programming a Delegate in SIMPL#

Starting with our ending SIMPL# code form the first document (Appendix A) there are a few steps to using a delegate we will need to go through to add in to our program.

The hierarchy of the SIMPL# code we are adding in looks like the following;

* Delegate Method
* Class for testing the delegate
  + Delegate Property
  + Method to do something with the Delegate Property
    - Call to the Delegate Property as a Method



*The editor allows you to collapse sections of code to make it easier to read. I used the minus sign on the left side to collapse Class1.*

**‘public delegate ushort MyDelegatesNameIsDave();’** – Declare a method (methods have the “();” after their names) with the ‘delegate’ type in your namespace, but outside of any class in your SIMPL# program. Make it public so SIMPL+ can see it and finally, give it a return data type.

**‘public class DelegateDavesClass’** – this is the class we will use to test the delegate. SIMPL+ needs to point to a public class in the SIMPL# code.

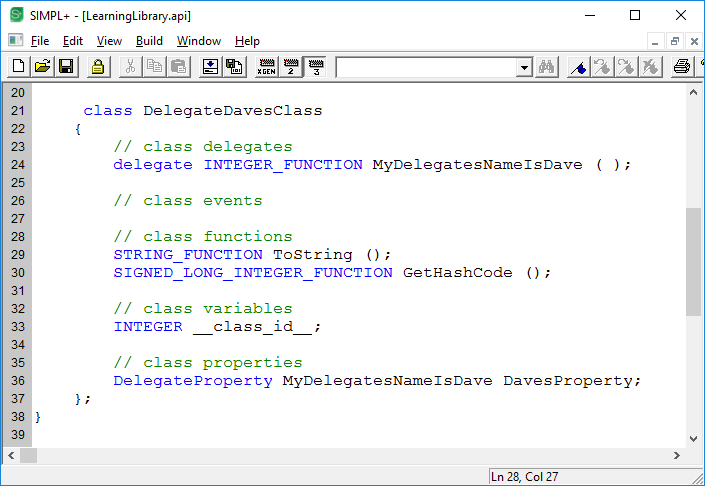
**‘public MyDelegatesNameIsDave DavesProperty {get;set;}’** – the delegate property…this is essentially a variable of the ‘MyDelegatesNameIsDave’ type that can hold data. [Simplified] MyDelegatesNameIsDave is of the ushort data type, so the property is of the ushort data type.

*One big difference between variables and properties is that with properties you can choose if they are read only, write only or read/write using ‘get;’ and ‘set;’ functions.*

**‘private ushort GetDavesPropertyValueFromSPlus()’** – this is a method within your delegate class that has the logic you create to work with the delegate property. When this method is called it will check the value of the delegate property. If the delegate property is not empty (null) we will run the delegate property as a delegate method (described below).

**‘ret = DavesProperty();’** – this expression causes the delegate method ‘DavesProperty();’ to run and assigns the output from it to our local variable ‘ret’ for storage.

Build the project in VisualStudio to update your .clz. Returning to SIMPL+ and checking the API again (Right-click in the SIMPL+ editor and select ‘Open API for LearningLibrary…’) will still have everything from Class1, but will show additional items within the new class ‘DelegateDavesClass’;

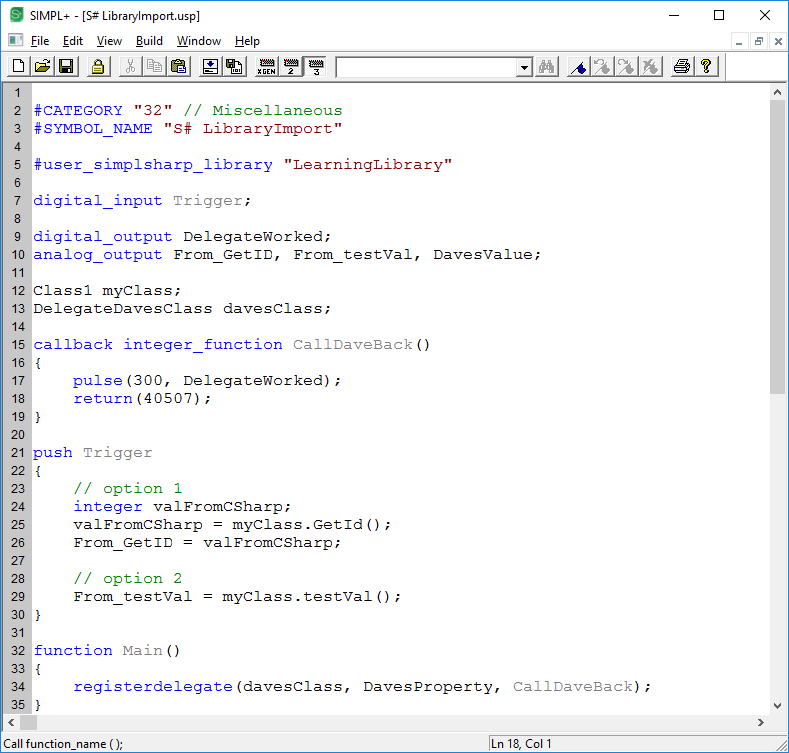


Programming (registering to) a Delegate in SIMPL+

SIMPL+ is able to access and run SIMPL# methods by creating an ‘object of the class’ and using the <class object>.<method name>(); format (see Introduction document for details).

Since there are no class definitions in SIMPL+, SIMPL# cannot create an object of anything in SIMPL+. To allow SIMPL# to access SIMPL+ functions the SIMPL+ program must first be pointed to the SIMPL# delegate through a process called registration (normally completed in the Function Main() section). Running a delegate method in SIMPL# causes the registered SIMPL+ code to act on the request.

Starting with our ending SIMPL# code from the first document (Appendix A) there are a few steps to using a delegate we will need to go through to add in to our SIMPL+ code;



**‘digital\_output DelegateWorked’** – we needed something for the SIMPL+ function to do once the SIMPL# code asked it to run.

**‘DelegateDavesClass davesClass;’** – instantiate an object of the class…ie. allow SIMPL+ to access methods in a SIMPL# class.

**‘callback integer\_function CallDaveBack()’** – this is the function that will run in SIMPL+ when the SIMPL# delegate property is accessed. Callback is required to denote this. The line ‘return(40507);’ means that the function CallDaveBack() will pass back the value of 40507 to whomever asked the function to run from the SIMPL# side.

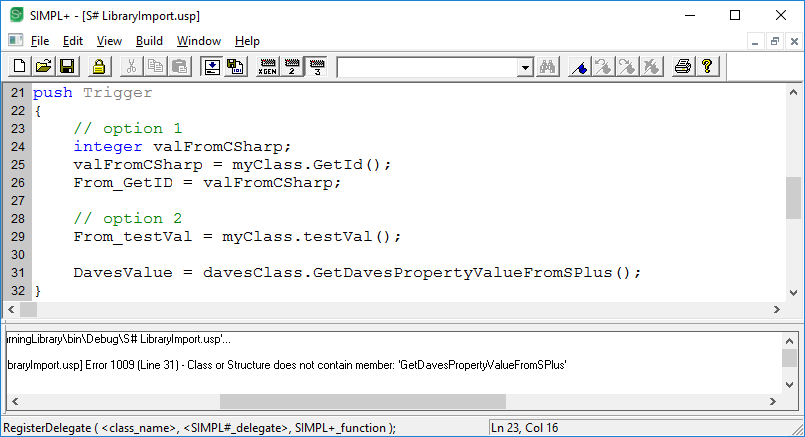
**‘function Main()’** – this is a special function existing at the bottom of the SIMPL+ code that runs once on program start…use this area to initialize variables or complete admin tasks in the program like registrations.

**‘registerdelegate(davesClass, DavesProperty, CallDaveBack);’** – this is the expression required to hook SIMPL# and SIMPL+ together to allow SIMPL# to run callback functions in SIMPL+. The 3 parameters need to be;

1. the object of the class (object name in SIMPL+) containing the delegate property in SIMPL#
2. the name of the delegate property from SIMPL#
3. the name of the callback function in SIMPL+ that will be run

Now, if the method GetDavesPropertyValueFromSPlus() is run from within SIMPL#, an output will pulse for 3 seconds and the delegate property in SIMPL# will have its value set to 40507!

BUT…with the code we made, we have no way of triggering a ‘private’ method in a SIMPL# class to test this out!!! See the compiler error below…SIMPL+ can’t see the method!



Using a ‘private’ declaration protects sections of code from access, like in this case from SIMPL+.

I will now change the declaration in SIMPL# from private to public and see what happens.

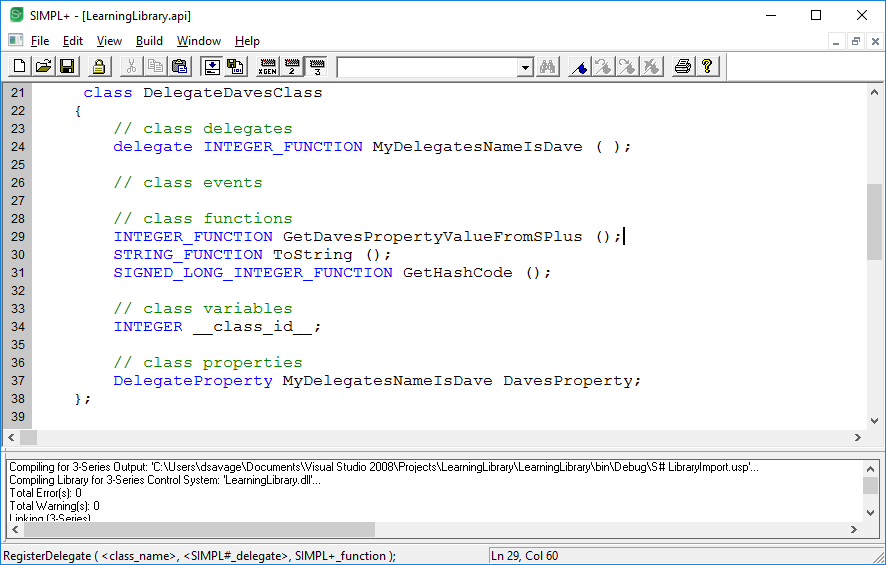
**‘private ushort GetDavesPropertyValueFromSPlus()’**

becomes

**‘public ushort GetDavesPropertyValueFromSPlus()’**

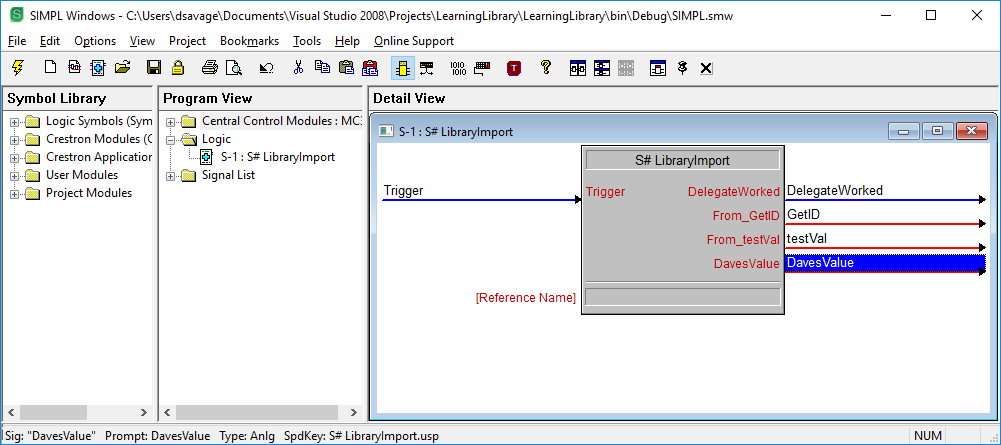
Build the VisualStudio project.

First thing we see is that the API in SIMPL+ shows the new method;

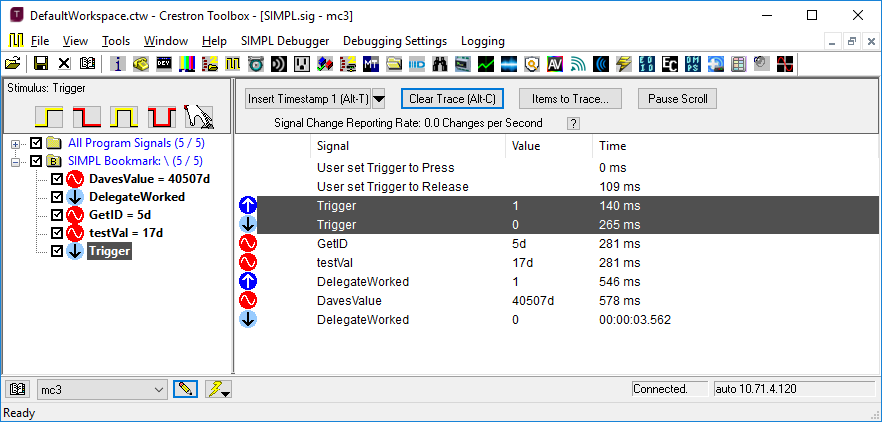


SIMPL+ should now compile without error.

Update your module signals in SIMPL, compile run and test.



Debugger Output



## So, what are Events used for in SIMPL#?

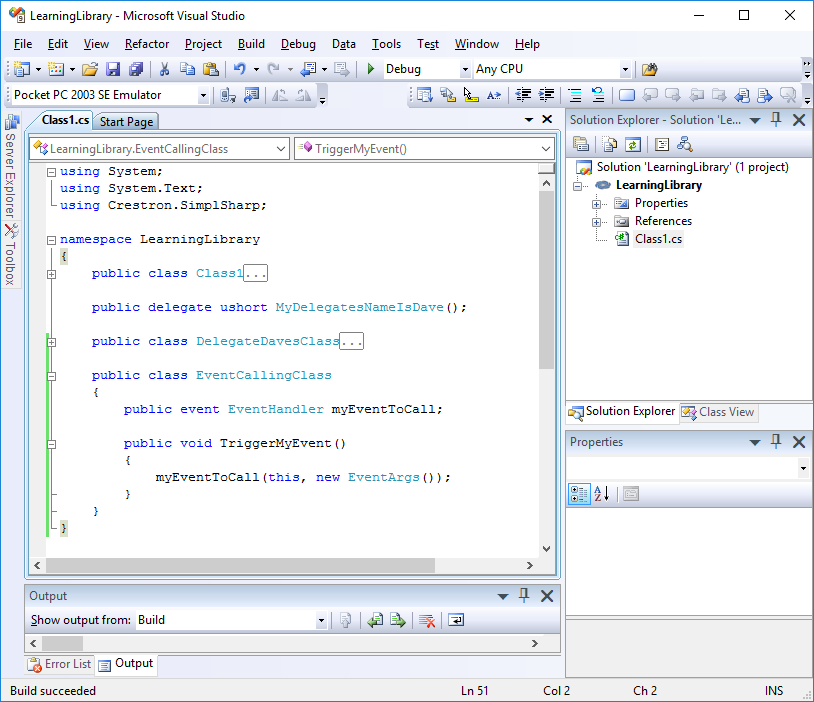
Events are used for allowing SIMPL# to make things happen in SIMPL+. Yes, it works just like a delegate!

So what’s the difference?

A delegate can contain many methods within the delegate class, in fact, you define all the methods contained within your delegate type. A single delegate can have multiple registrations in SIMPL+ pointing at it to access all of the various methods that could be inside the SIMPL# class that contains the delegate property.

An event is a much less complicated delegate and is designed to signal other parts of the SIMPL# code or SIMPL+ that something specific has happened.

## Create the event in SIMPL#



**‘public class EventCallingClass’** – I’ve created a standalone class to test the process of ‘raising/triggering an event’. An Event can be part of another class mixed with other code as well.

**‘public event EventHandler myEventToCall;’** – here I’ve essentially created a variable (called a member) of the EventHandler data type. Adding in the declaration ‘event’ will allow me to run this member as an event method.

This is easier than how a delegate does this; declare the delegate method, declare a property of the delegate method type, then run the property like a method to access the get/set options.

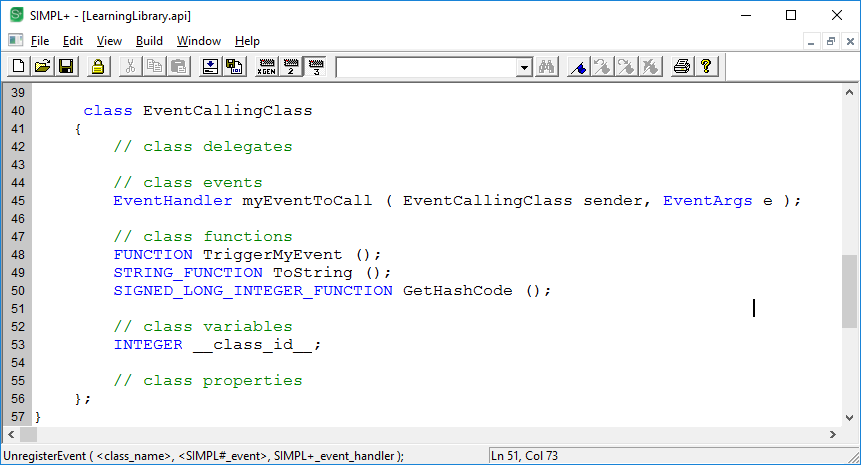
**‘public void TriggerMyEvent()’** – this method is the one we can run from other places in the SIMPL# code or even from SIMPL+ to make our event occur. ‘void’ means it does not give back any data (does not return a value back to whomever called it).

**‘myEventToCall(this, new EventArgs());’** – this expression in our method is what raises the event. In SIMPL+ we will register to the Event Handler and create an Event Handler function. The SIMPL+ Event Handler function is the code in SIMPL+ that runs when the SIMPL# method raises the event.

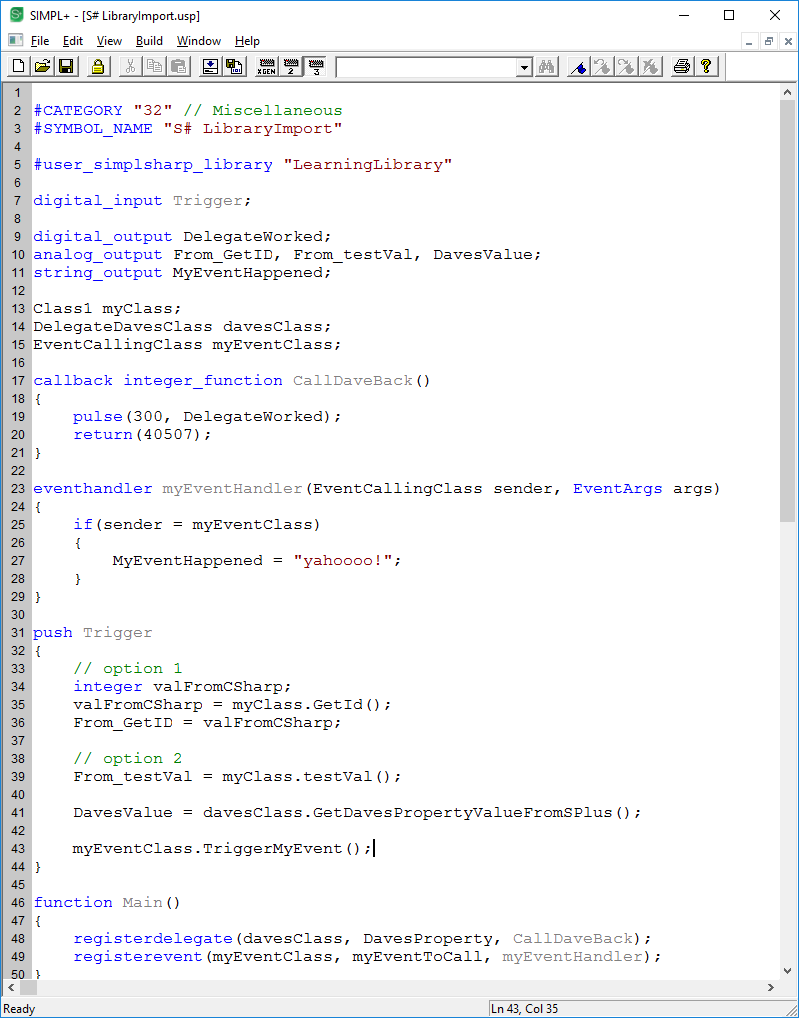
This method has 2 parameters in brackets. ‘this’ is the information about the class that is sending the notification that the event was raised. This is called the sender. When using the keyword ‘this’ it means the class that the called event is in (EventCallingClass). ‘new EventArgs()’ is an object of the EventArgs class in C#. Yes, we are passing a reference to this class along to whoever is registered to this event in SIMPL+.

## Register to the Event in SIMPL+

Your API for LearningLibrary will now show the following additions;



Notice that TriggerMyEvent does not have a data type? When it was declared in SIMPL# is was given the return type of void.

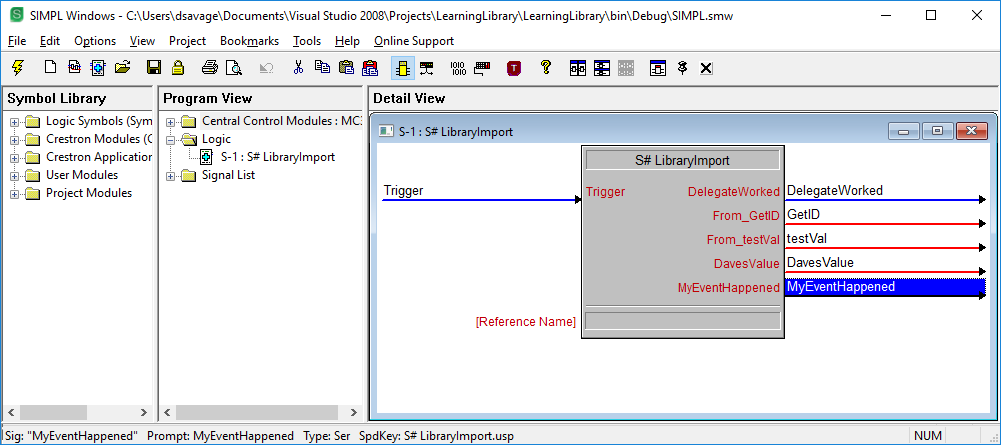


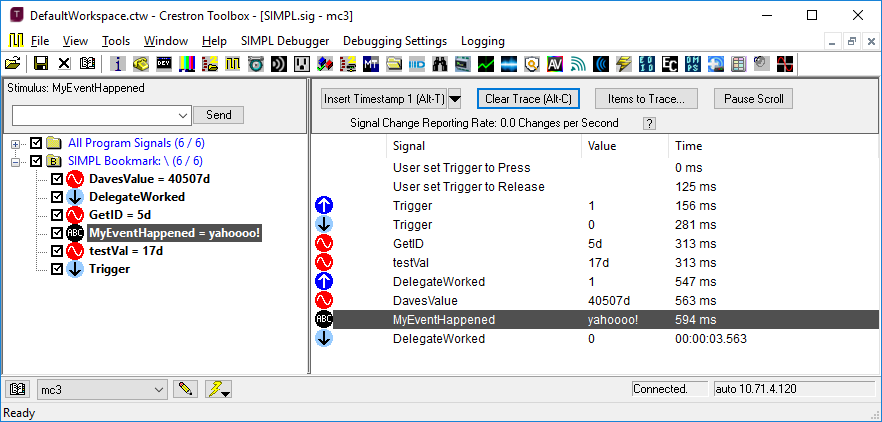
**‘string\_output MyEventHappened’** – I needed to get SIMPL+ to do something once the event was raised in SIMPL#.

**‘EventCallingClass myEventClass;’** – this is an object of the SIMPL# class that contains the event. This also allows SIMPL+ to access all public methods and members inside the SIMPL# class. This line is needed so SIMPL+ can register to the event.

**‘eventhandler myEventHandler(EventCallingClass sender, EventArgs args)’** – this is the function that will run in SIMPL+ when the event is raised in SIMPL#. Since the called method in SIMPL# (myEventToCall(this, new EventArgs());) is passing 2 parameters along, the registered function in SIMPL+ must have the same number of parameters.

In the background, the Event actually has a hidden private delegate that is forcing you to match the number of parameters on the event, even if you don’t use the passed in information. This is called the ‘signature’ of the delegate.





# Appendix A

## STARTING SIMPL# CODE

using System;

using System.Text;

using Crestron.SimplSharp;

namespace LearningLibrary

{

public class Class1

{

ushort idNum;

public Class1() // default constructor

{

}

public ushort GetId()

{

idNum = 5;

return idNum;

}

public ushort testVal()

{

ushort myVal = 17;

return myVal;

}

}

}

## STARTING SIMPL+ CODE

#CATEGORY "32" // Miscellaneous

#SYMBOL\_NAME "S# LibraryImport"

#user\_simplsharp\_library "LearningLibrary"

digital\_input Trigger;

analog\_output From\_GetID, From\_testVal;

Class1 myClass;

push Trigger

{

// option 1

integer valFromCSharp;

valFromCSharp = myClass.GetId();

From\_GetID = valFromCSharp;

// option 2

From\_testVal = myClass.testVal();

}