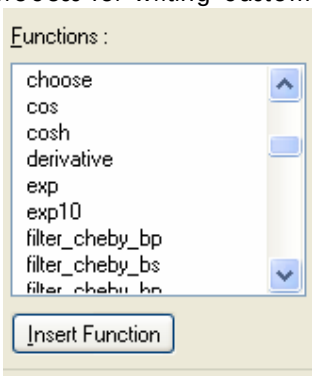
Document Number		DTN0008	
Title		Creating an i2 Math Function	
Approved By		*AN*	
Revision	**Date**	**Prepared By**	**Change History**
1.1	23/6/2006	JA	

Introduction

MoTeC's i2 application sets a new standard in data analysis software. Included within i2 is an extensible maths engine that lets you manipulate your data with a large number of built in maths functions. Although the list of maths functions provided with i2 is comprehensive, sometimes analysis of your data requires specialised functions. This document outlines the process for writing custom math functions that can be used in i2 Pro projects.



Scope

This document applies to i2 Pro

Math Functions

MoTeC provides a template that can be used to create a new maths function. It is written externally of i2 using VB, C++ or another programming language. Once created, they can be added to the functions list as shown, and can be called just like any other maths function within an i2 math expression.

An example of a possible function will be used to help explain the process:

For example: you may want to create a specialised status hold channel that is active whenever the engine RPM is above 5000rpm for 2 seconds and inactive the rest of the time.

Software Required

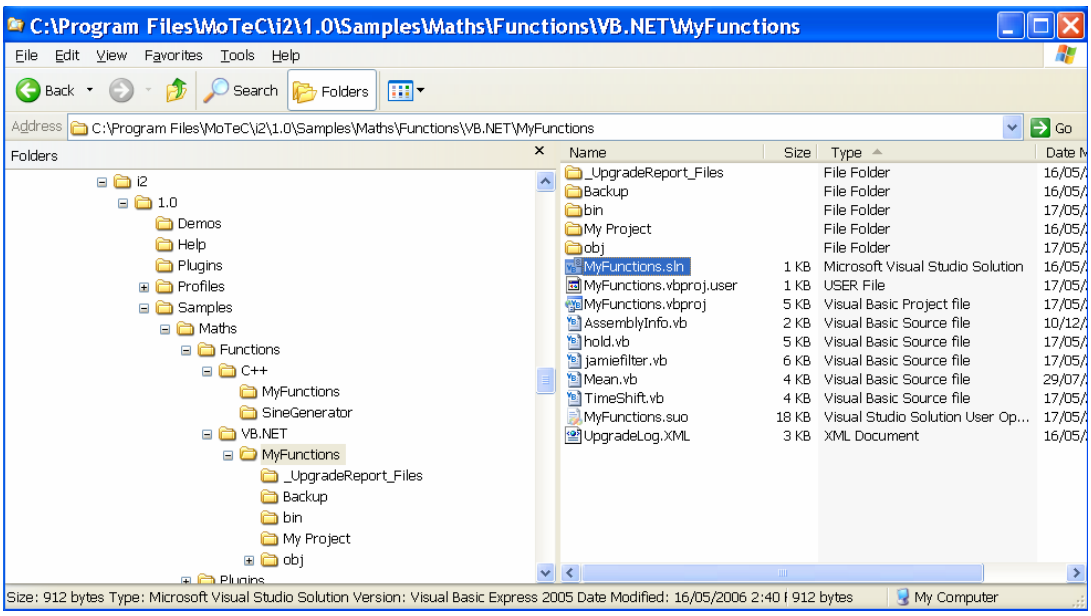
To assist in creating a new function, MoTeC provides some samples with i2 Pro that are written in VB.net and C++. You will need a general understanding of programming in either of these languages to complete this process, as well as the appropriate VB.net or C++ programming software. VB.net is available for free from the Microsoft web site in an express edition, but if you were to do maths function programming regularly, it would be useful to buy a full VB.net version, as some of the reduced functionality in "VB.net express" makes the job far more time consuming.

VB.net example

To start, decide whether you are going to use VB.net or C++, then navigate to the appropriate sample directory. In this example we will use VB.net, the samples for which are in:

"C:\Program Files\MoTeC\i2\1.0\Samples\Maths\Functions\VB.NET\MyFunctions".

The files there include the VB.net solution file (.sln), and the VB.net source files. Start by double clicking on the solution file "myfunctions.sln" to open it within VB.net.



Location of the VB functions directory.

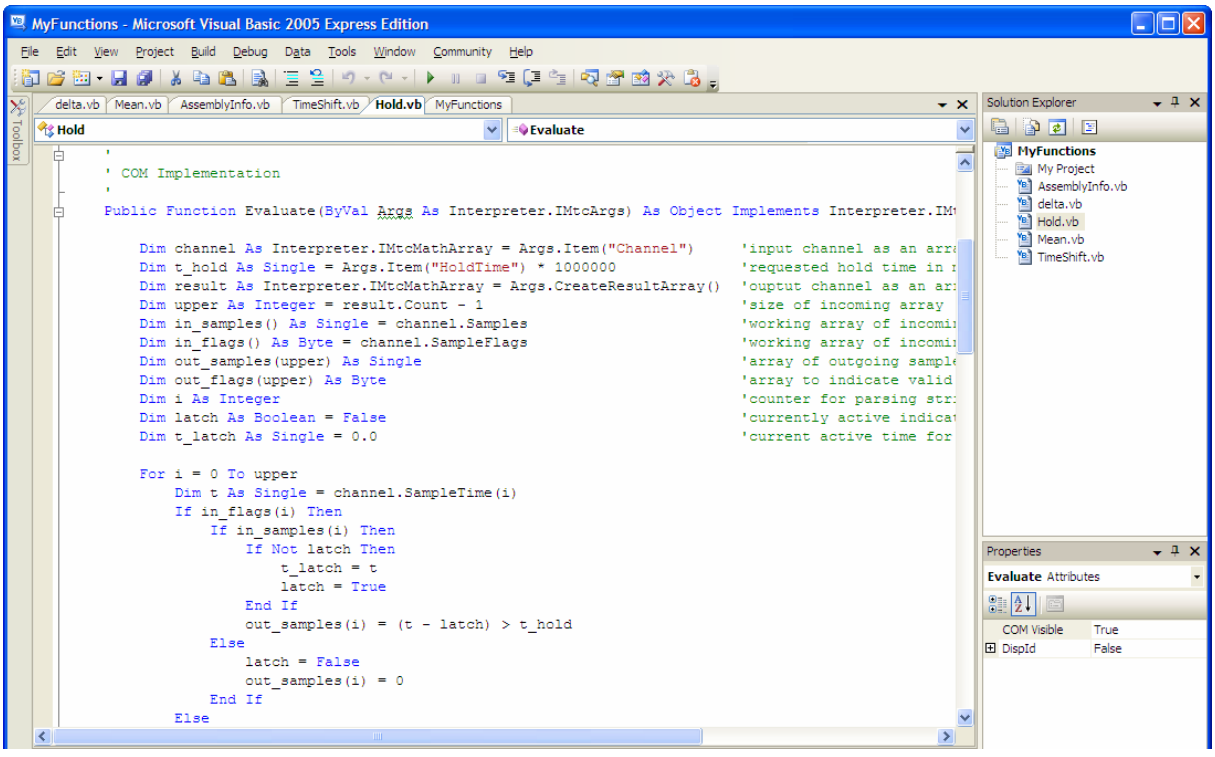
The picture below shows the "MyFucntions" solution file and VB source code files open in Visual Basic 2005 Express. The code is written in such a way that you should be able to leave the majority of the structure and code in place and modify a copy of an existing function to suit your requirement. We will begin by using an existing function to form the basis of our new function.

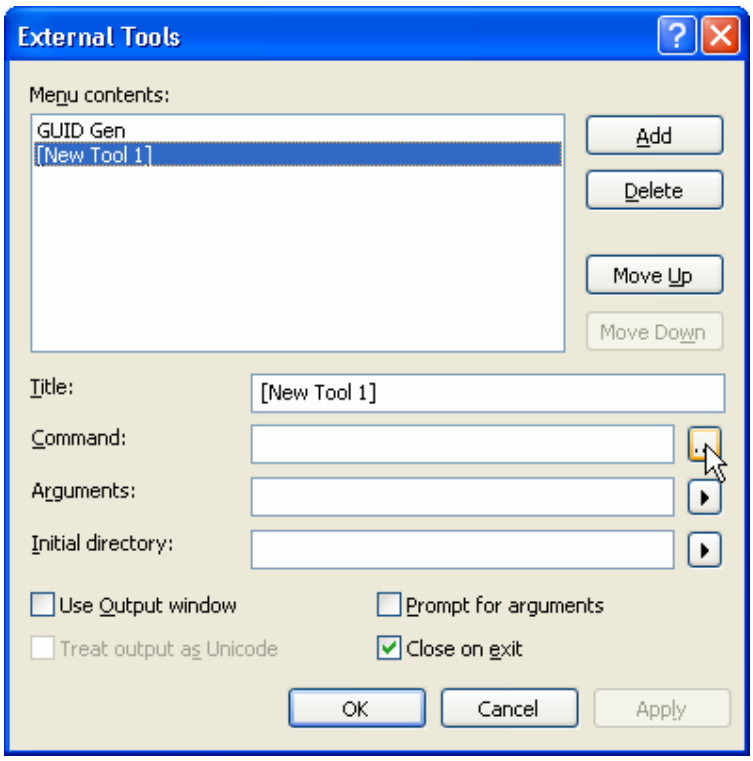
- Goto 'Project – Add New Item' and select the icon 'Class', enter the name "Hold.vb"

- Select all of the code in 'TimeShift.vb' and copy it into 'Hold.vb'

- Replace all instances of the text 'TimeShift' with 'Hold'

- new GUIDs will need to be generated for this function. Go to 'Tools – External Tools' and click on 'Add', then the select button next to the 'Command' parameter





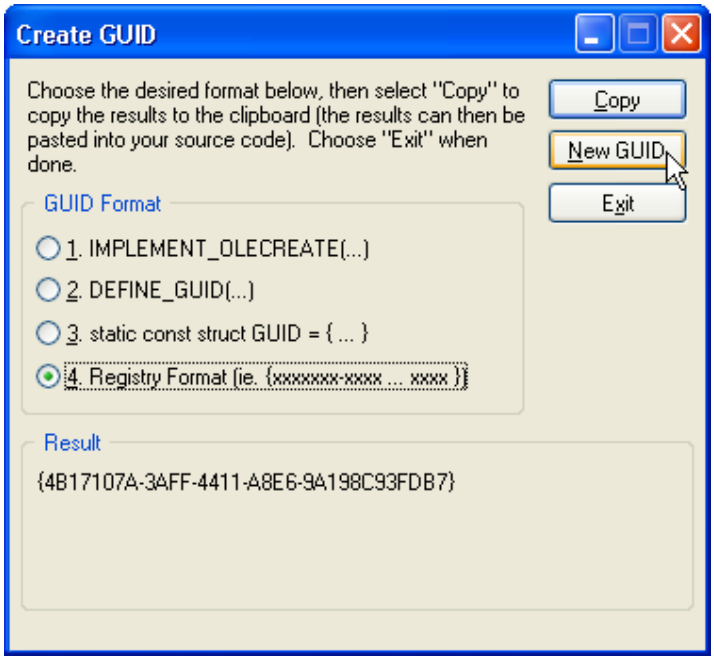
- Navigate to the directory 'Common7 – Tools' and open the file 'guidgen.exe'

- Change the title from "[New Tool 1]" to "GUID Gen" and click on OK.

- Note that once installed, the GUID Gen tool can then be used to generate IDs for any further new Functions.

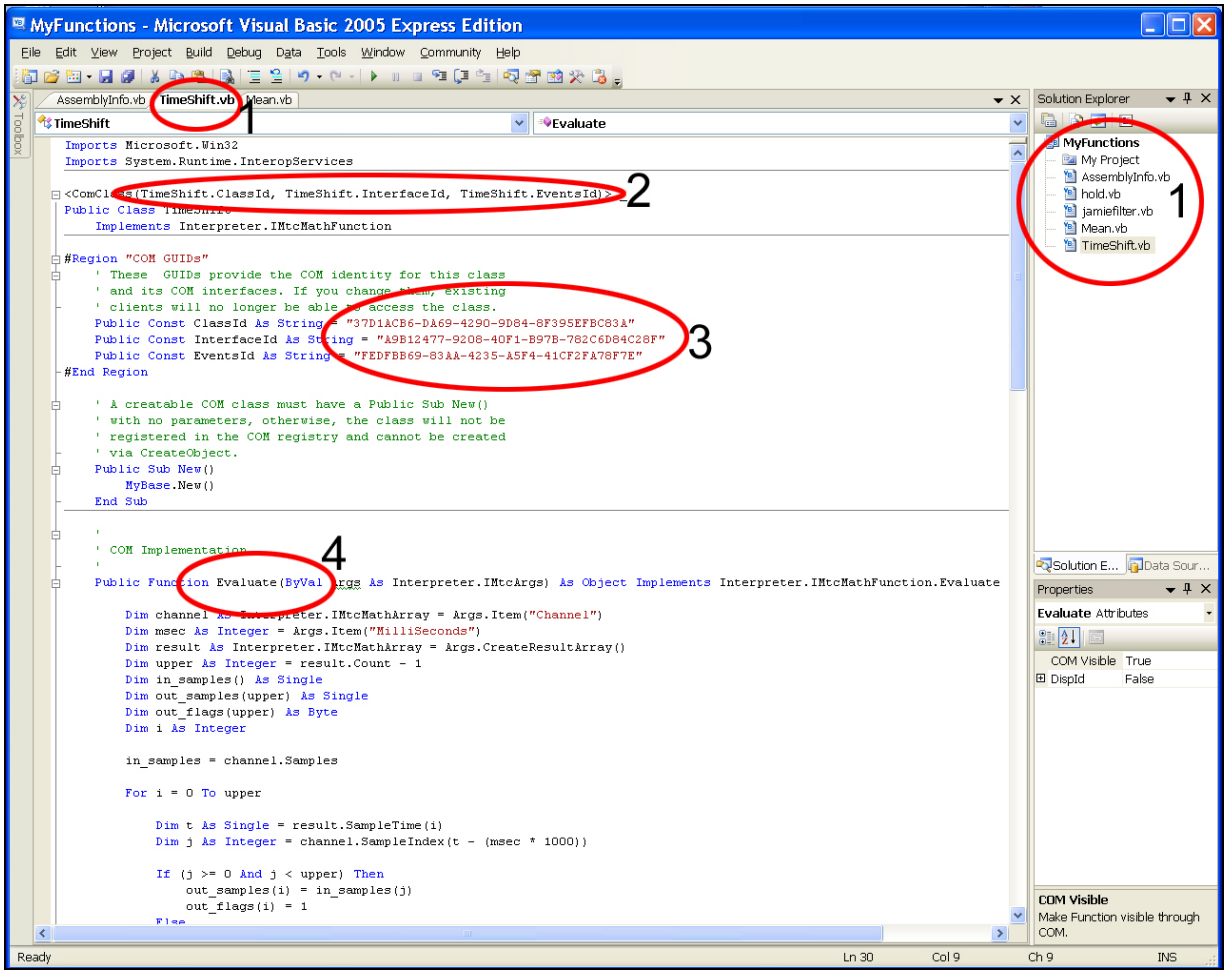
- We need to generate 3 new GUIDs for the constants in our new function. Under the 'Tools' menu there should now be an item "GUID Gen", click on this to run the GUID generator.

- Select GUID Format number 4 "Registry format" and click on 'New GUID'



- Copy the result to the clipboard and then paste this into the String for 'ClassId', removing the brackets.

- Repeat for 'InterfaceId' and 'EventsId'



Features of the TimeShift function

1) is the name of the class, in this instance it is Timeshift – our new Function is 'Hold'

2) shows the place where you need to modify the ids to refer to "hold" instead of timeshift.

3) Shows where the 3 news COM GUIDs we generated are to be copied into the Hold function

4) Evaluate is where you actually code your function. You can use the 'timeshift' or 'mean' example to see how an example works,

RegisterFunction

This section of the code sets the registration details for our new function, and is where we define what is being passed to and from i2. First set the name of the input array (ie: source channel) within VB, what sort of data is it going to be and what sort of output channel it will be. You can add more than one array as an input, as well as accepting in multiple single values (constants) as well.

The declarations here can be modified to suit your function. In our example, the function is going to have two inputs. One: a channel array which we intend to use for RPM, but to keep the function flexible we will just call it "channel". Two: a single value called "holdtime". In the Evaluate Function code the input array is referred to as "channel", and the input value as "holdtime".

hold (Declarations)

```
COM Registration

omRegisterFunction()> _
red Sub RegisterFunction(ByVal t As Type)

  Dim key As RegistryKey
  Dim args_key As RegistryKey
  Dim arg_key As RegistryKey
  Dim return_key As RegistryKey

  key = Microsoft.Win32.Registry.LocalMachine.CreateSubKey("SOFTWARE\\MoTeC\\Analysis\\1.0\\Maths\\Functions\\" + t.Fu
  key.SetValue(Nothing, "MyFunctions.hold")                              'funciton name
  key.SetValue("Description", "creates a status channel indicating when channel is active for more than n seconds")
  key.SetValue("Flags", 0)

  ' Arguments
  args_key = key.CreateSubKey("Arguments")                               'number of input arguments
  args_key.SetValue(Nothing, 2)

  arg_key = args_key.CreateSubKey("0")                                   'input key
  arg_key.SetValue(Nothing, "Channel")                                   'input name
  arg_key.SetValue("DataType", Interpreter.DataType.Real)                'input data type
  arg_key.SetValue("Dimension", CInt(Interpreter.Dimension.Array))       'input dimension type

  arg_key = args_key.CreateSubKey("1")                                   'input key
  arg_key.SetValue(Nothing, "holdtime")                                  'input name
  arg_key.SetValue("DataType", Interpreter.DataType.Integer)             'input data type
  arg_key.SetValue("Dimension", CInt(Interpreter.Dimension.Scalar))      'input dimension type

  ' Return Value
  return_key = key.CreateSubKey("Returns")                               'output name
  return_key.SetValue("DataType", Interpreter.DataType.Real)             'output data type
  return_key.SetValue("Dimension", CInt(Interpreter.Dimension.Array))    'output dimension type

d Sub
```

RegisterFunction code in Hold.vb

In the example above, the input array is called channel, is a real data type and is an array of values. Next is “Holdtime”, an integer which is a scalar or single value. Finally there is the return value, it is called “Returns”, that is an array of real data type values. This information is added to the registry settings of the PC, and used by i2 to setup the function and its details. There are also settings here to set the function name, “Hold”, and a description of the function, as seen in i2 when you select the function in i2 "Creates a status channel indicating when a channel is active for more than n seconds”)

Evaluate

Now that we have set all that up, we are ready to code our function.

```
Public Function Evaluate(ByVal Args As Interpreter.IMtcArgs) As Object Implements Interpreter.IMtcMathFunction.Evaluate


    Dim channel As Interpreter.IMtcMathArray = Args.Item("Channel") 'the array of data incoming
    Dim t_hold As Single = Args.Item("HoldTime") * 1000000          'the requested time in microsecs to t_hold
    Dim result As Interpreter.IMtcMathArray = Args.CreateResultArray()  'output array
    Dim upper As Integer = result.Count - 1                         'the size of the incoming array
    Dim in_samples() As Single = channel.Samples                    'working array of incoming samples
    Dim in_flags() As Byte = channel.SampleFlags                    'working array of incoming samples flags
    Dim out_samples(upper) As Single                                'array of outgoing samples
    Dim out_flags(upper) As Byte                                    'array to indicate valid sample
    Dim i As Integer                                                'counter for parsing string
    Dim latched As Boolean = False                                  'Currently active indicator
    Dim t_latch As Single = 0.0                                     'current latched time

    For i = 0 To upper                                              'start loop for array size
        Dim t As Single = channel.SampleTime(i)                     'set T as current time in array in microseconds
        If in_flags(i) Then                                         'If we have a valid sample
            If in_samples(i) Then                                   'and it's on/true
                If Not latched Then                                 'if start of new time period
                    t_latch = t                                     'set section time to current time
                    latched = True                                  'set latched to true
                End If
                out_samples(i) = (t - t_latch) > t_hold             'if it's been latched longer than request, set output true
            Else
                latched = False                                     'not a valid sample
                out_samples(i) = 0                                  'output sample is set to false
            End If
        Else
            latched = False                                         'input status false, so set latched off
            out_samples(i) = 0                                      'output sample is set to false
        End If
        out_flags(i) = in_flags(i)                                  'set output flag(status) to match input array

    Next i

    result.Samples = out_samples                                    'copy the newly created output values to output array
    result.SampleFlags = out_flags                                  'copy the newly created output flags to output array

    Return result

End Function
```

code for the "hold" function

From a coding perspective, it is just normal VB code, but to help understand, here is a short explanation.

Channels input array example

Array position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Channel input Values	1	0	0	0	0	1	0	0	1	1	1	1	1	1	0

Flags input array example.

Top row input values, bottom row array position

Array position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Flag input Values	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

The value from i2 as set in the maths function, input 2

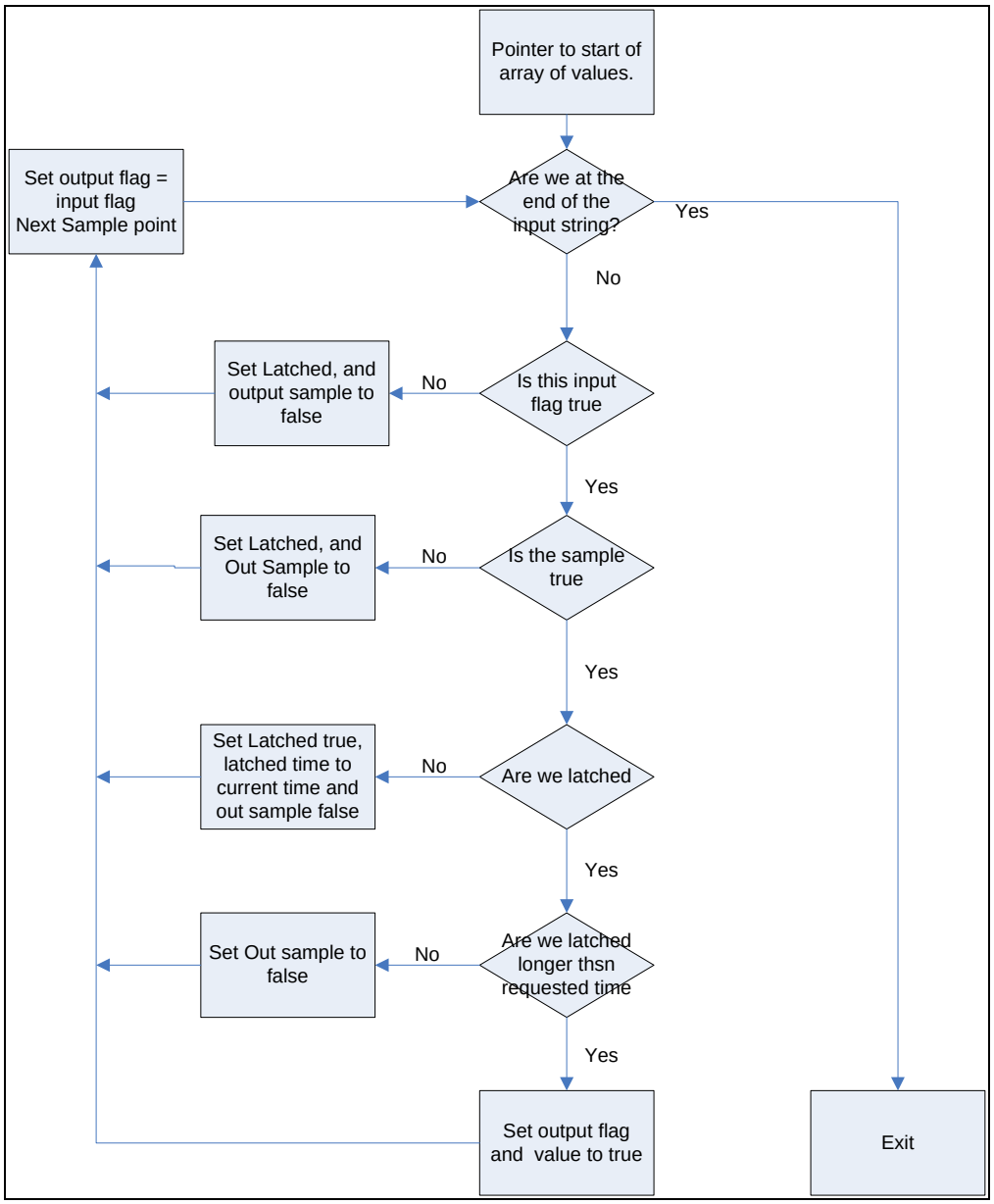
Holdtime	2

So for this function example, we are parsing the Channels input array, looking for "1" (true) values. If we see "1", then set the start time. Next time, if it is still 1, then check if it is longer than the

requested hold time. If it is turn the output array at that position on, else turn it off. If the input array is a "0", set the time back to 0.

If you want the current value to be displayed in i2, you must set the value in the output *flag array* to "1" at the same position as the value in the channel values array. This mechanism is used to tell i2 that the data at that sample is valid.

For example, in the sample array above, none of the values would be displayed in i2, except for the one at array position 11. The other data points would be shown in i2 as "n/a". To show a value of '0' then the flag array must be set to 1 and the data to 0.



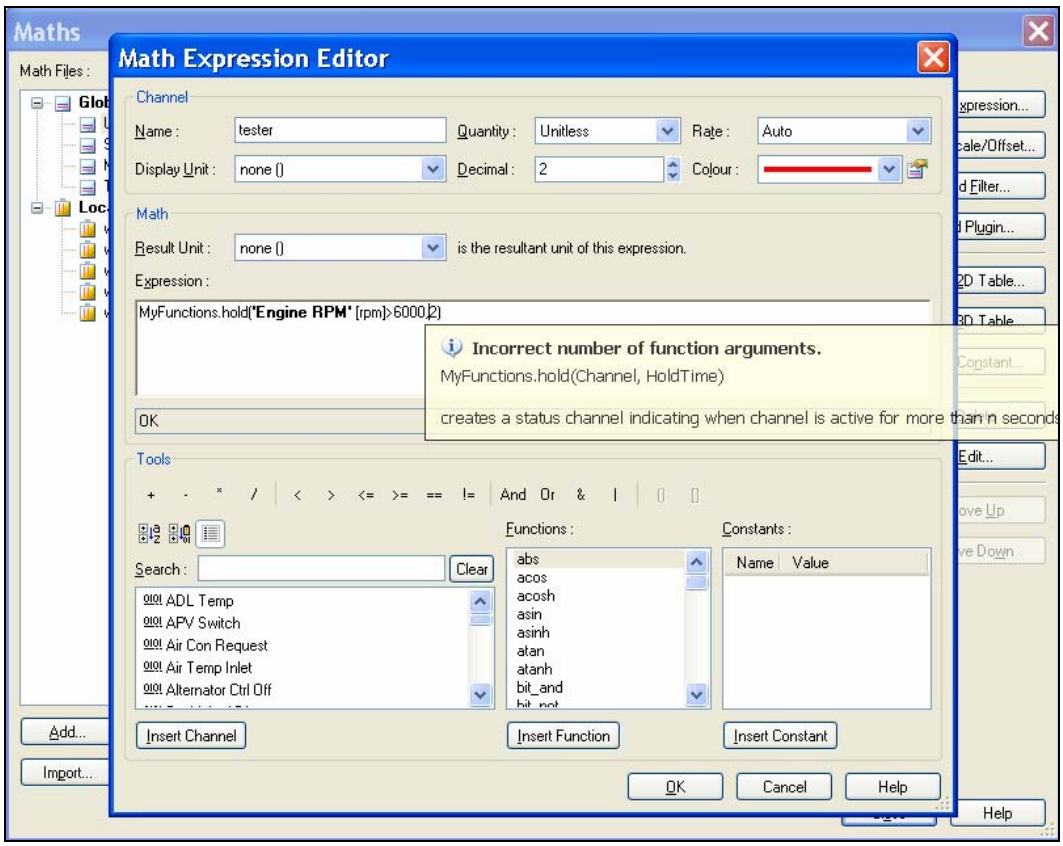
Hold function Flowchart

It is also important to know that if you refer to a sample time, it is returned as a value that is 1 1000000th of a second, so you will need to multiply your input by 1000 if you want to refer to it in milliseconds, or 1,000,000 if you want to refer to it in seconds.

Debugging

Go to menu – 'Build - Build MyFunctions'. All being well, the new function has now been registered and can be used. Now after re-starting i2 Pro, the function you have just written will be available in the function list as MyFunctions.hold

Below is an example of the "MyFunctions.Hold" function in an i2 math expression. The function template that appears uses the Function Arguments and the text that was entered as the Description in the RegisterFunction code.



How the finished function would appear in i2

If you use the free express version of VB.net, one of the downsides is that you can't use VB to start i2 to run your code in debug mode. You need to purchase a full version of VB.net to do this. So the only way to debug your code is by using MSGBOXs that pop up in i2 after you have built your code and when it tries to execute it (the first time it has to display the data created by your code). The MSGBOX commands throughout the code are for debugging purposes.

So if the code fails for some reason, you get no error message, it just doesn't complete and you get no data you need to use MSGBOXs throughout the code to trace its execution and see where the code it is failing. It would be far more efficient to use the full version of VB, but it can still be done with the free version.