Document Number		DTN0009	
Title		Creating an i2 Math Plugin	
Approved By		*AN*	
Revision	**Date**	**Prepared By**	**Change History**
1.1	23/6/2006	JA	
1.2	18/08/2014	PC	Note added on action to take if plugin does not appear

Introduction

MoTeC's new i2 application sets a new standard in data analysis. Included within i2 Pro is an extensible maths engine that lets you manipulate your data with a large number of built in maths functions.

To further extend the data analysis capability of i2, MoTeC have provided a user plugin system. This allows a user to export whole channels from i2, then perform calculations within an external program, eg: one written in VB.net. The results of any calculations can be output to external files or even sent back into i2 as a new channel for analysis.

How to use the Plugin system with i2 Pro

This document should be read after reading MoTeC Technical Note "DTN0008 Creating an i2 Math Function" as many of the concepts and methods are explained in detail there first.

Provided with i2 Pro is a sample file with examples of a few of the many features available within the i2 plugin system. This sample file can be found in the directory:

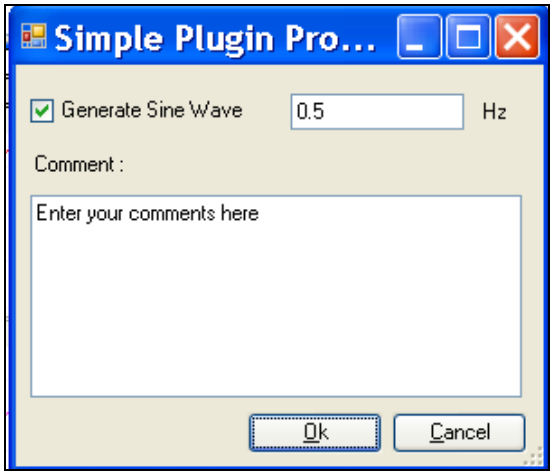
"C:\Program Files\MoTeC\i2\1.0\Samples\Maths\Plugins\VB.NET\MyPlugins"

If you open the "MyPlugins.sln" solution file you will find the sample application, "Simple.vb". Read through this program for a feel for how plugins work, then build the application to make it available within i2. This plugin is used to create a channel that consists of a sine wave with a frequency selected by the user.

After a plugin has been built, it needs to be installed in i2 by going into the maths editor, and selecting "Add Plugin". From here you can select from all built plugins in the system. In this case you should see "MyPlugins.Simple", select this and click on OK.

Note: If a built plugin does not appear, verify that i2 is running with administrator permissions (in some cases Windows User Account Control must also be disabled). Also ensure that the plugin dll file that you want to install is located in the same folder as Interop.i2.dll.

Upon adding, a properties box opens up, and you can set the options that you want the user to define for your plugin. In this example, there is a comment area, a tick box to say if you want to generate the sine wave or not and the sine wave frequency.



simple plugin dialog

Then when the maths window is closed, the plugin will run. A number of dialogs will open up with details that the plugin has grabbed from the current main log file details. Then several new channels will be available, including 'Simple sine Wave'.

Create your own Plugin.

In this example we will create a new plugin that is going to write an output file of shock travel data from i2 to a text file that can be sent to your damper dyno to simulate the race or event. The input to this will be a damper channel. Within VB we will add to the damper position array a sample time so that the damper dyno can work out how fast to change position. The output will be damper position to 3 decimal places, then time in seconds to 3 decimal places, then a new line. There will be one row per sample point in the selected file.

```
Option Explicit On
Imports Microsoft.Win32
Imports System.Runtime.InteropServices
Imports System.IO
<ComClass(outfile.ClassId, outfile.InterfaceId, outfile.EventsId)> _
Public Class outfile
    Implements Interpreter.IMtcMathPlugin

#Region "COM GUIDs"
    ' These  GUIDs provide the COM identity for this class
    ' and its COM interfaces. If you change them, existing
    ' clients will no longer be able to access the class.
    Public Const ClassId As String = "473DFCAB-6175-4e7e-97AA-2B536F606070"
    Public Const InterfaceId As String = "8A5EA3CF-25C5-4b5b-BA07-57D9109DE690"
    Public Const EventsId As String = "DC363627-40D2-4b83-83F6-CFD9CCBCBC9D"
#End Region

    ' A creatable COM class must have a Public Sub New()
    ' with no parameters, otherwise, the class will not be
    ' registered in the COM registry and cannot be created
    ' via CreateObject.
    Public Sub New()
        MyBase.New()
        m_Sine_Generate = True
        m_Sine_Frequency = 0.5
        m_Sine_SampleRate = 20.0
        m_Comment = "Enter channel name to output here"
    End Sub

    '
    ' This implements the MoTeC Math Plugin Interface
    '
    Dim m_Comment As String
    Dim m_Sine_Generate As Boolean
    Dim m_Sine_Frequency As Single
    Dim m_Sine_SampleRate As Single

    Public ReadOnly Property InputChannels() As System.Array Implements Interpreter.IMtcMathPlugin.InputChannels
        'sets up for the input file to be recieved from i2
        Get
            Dim Inputs(1) As String
            Inputs(0) = "Damper position"
            Return Inputs
        End Get
    End Property
```

'outfile' code

Here is the first page of the damper dyno output plugin that called outfile. To start this file, copy the provided Plugin sample file, Simple.VB to a new class called outfile.vb. Follow the process used in the functions example to change all references of simple.vb to outfile, and used the GUIDGEN application to create unique GUIDs for this plugin.

You can see above that much of the original settings page as provided in the sample remains, as much is still used here. The "m_sine_generate", "frequency", and "sample rate" are not used, the the m_comment is used as the source for the channel name of the damper for the output file. The default values can be set in the 'New' subroutine. The definitions of the plugin interface variables are set just below this. Further below that is the setup of the channel/channels to be received from i2. Here you define the number of channels to be received and the name of each one as it will be referred to within VB.

```
    Public ReadOnly Property OutputChannels() As System.Array Implements Interpreter.IMtcMathPlugin.OutputChannels
        'Sets up the file to be sent into i2.
        Get
            Dim Outputs(1) As String
            Outputs(0) = "outfile Damper Pos"
            Return Outputs
        End Get
    End Property
    Public ReadOnly Property Summary() As String Implements Interpreter.IMtcMathPlugin.Summary
        Get
            Return "This is my sample plugin. It generates a fiele containing damper position and time channels fo
        End Get
    End Property

    Public Property Settings() As String Implements Interpreter.IMtcMathPlugin.Settings
        'gets settings each time the plugin is called
        Get                                     'goes and retrieves the most recent value of m_comment
            Return m_Comment
        End Get
        Set(ByVal Value As String)      'Sets the m_comment channel to the value last set in the settings dialog
            m_Comment = Value
        End Set
    End Property
    Public Sub ShowSettings() Implements Interpreter.IMtcMathPlugin.ShowSettings

        'displays the settings box details only when you add or edit the plugin

        Dim dlg As New SimpleProps
        m_Comment = "Enter channel name to output here"
        dlg.TextBox1.Text = m_Comment
        If (dlg.ShowDialog() = System.Windows.Forms.DialogResult.OK) Then
            m_Comment = dlg.TextBox1.Text
        End If
        dlg.Dispose()

    End Sub
```

"outputchanels" return data to i2

In the "outputchannels" section of code, we set up the channels that are going to be sent back to i2. We need to define the output channel names and the number of outputs.

The Summary section includes a text string that indicates the purpose of this plugin.

The Settings section is run each time the plugin is called to get the values from the settings panel used within the plugin. The Set goes and retrieves the value of m_Comment as defined by the user in the settings dialog box.

The "ShowSettings" area is used to define the values to be displayed by default in the settings screen, then display the screen and assign the returned values to the m_ variables. This section refers to the "Simpleprops" design and code which gets all the settings for your plugin. You could use the "Simpleprops" as your settings box, and modify the screen, its layout, inputs and outputs here for your application. This same box is used in the sample "Simple.vb" plugin, so in reality, it would probably be best to build for yourself another input box to suit your new application.

```
    Public Sub Register(ByVal DataSource As Interpreter.IMtcDataSource) Implements Interpreter.IMtcMathPlugin.Regis
        'This section sets up the channel parameters for i2, such as trace colour, decimal places, scaling etc.
        On Error GoTo ErrorHandler
        Dim DamperPos As Interpreter.IMtcChannelDescriptor

        DamperPos = DataSource.Descriptors("outfile Damper Pos")
        If (DamperPos Is Nothing) Then
            DamperPos = DataSource.Descriptors.Add("outfile Damper Pos", "MoTeC.Float", "mm")
        End If
        DamperPos.ColorIndex = 1
        DamperPos.DPS = 3
        DamperPos.Interpolate = True
        DamperPos.ScaleMin = -50        ' mm
        DamperPos.ScaleMax = 50     ' mm
        Exit Sub
ErrorHandler:
        '   An error has occured if we reach this code. This error can be due :
        '   - Trying to access a NULL object
        '   - Attempting to call a method that does not exist
        Resume Next

    End Sub

    Public Sub Execute(ByVal DataSource As Interpreter.IMtcDataSource) Implements Interpreter.IMtcMathPlugin.Execut
        On Error Resume Next

        Dim normal As Interpreter.IMtcDataLayer = DataSource.Layers(Interpreter.Layer.Normal)

        ' This plugin takes an input channel name from the settings box, and outputs it at its sample
        ' rate to an output file called c:\testfile.txt for use in a damper dyno.

        Dim DamperPos As Interpreter.IMtcChannel = normal.Channels(m_Comment)
        Dim Dampersamples() As Single = DamperPos.DataArray.Samples
        Dim dampertime As Double = DamperPos.DataArray.Rate
        Dim upper As Integer = DamperPos.DataArray.Count - 1
        Dim currenttime As Double = 0
        Dim i As Integer

        dampertime = 1 / dampertime                                         'calculate sample time from rate
        Using sw As StreamWriter = New StreamWriter("c:\TestFile.txt")      'Add some text to the file.
            For i = 0 To upper                                              'loop for length of array
                sw.WriteLine(currenttime.ToString("F3") & " " & Dampersamples(i).ToString("F3")) 'format and output
                currenttime = currenttime + dampertime                      'increment sample time to next poir
            Next i                                                          'loop to next sample point
            sw.Close()                                                      'close file
        End Using
        normal.Channels("outfile Damper Pos").SetData(DamperPos.DataArray, Interpreter.ChannelStatus.Valid, "outfil
        Exit Sub

    End Sub
```

Register

In the Register section we set up the detailed channel properties for any channel that we are sending back into i2. We need to set the channel name, the type of channel, its units, colour, number of decimal places, the interpolation settings and the scaling settings. This must be done for each channel that we intend to send back to i2.

The Execute section, as per the Functions example, is where you actually code the Plugin. In this example we are retrieving the “DamperPos” values then writing them out to the local file one at a time. We are formatting the output values with a ‘ToString’ function that rounds the values to 3 decimal places (F3) in the output file. Down the bottom of the Subroutine, we are also outputting the “DamperPos” channel as “outfile Damper Pos” back to i2. This is useful so that within i2, you can see which channel you have exported to the file, and what the trace looks like.

Conclusion

This is a basic example of how plugins work, more complex functions are limited only by programming capability. Many more i2 specific functions are shown in the ‘Simple’ example plugin discussed earlier. MoTeC’s plugin and function systems make i2’s data analysis extremely powerful and customisation virtually limitless.