

# MapWindow Plug-in Development

## Sample Project: Simple Path Analyzer Plug-in

A step-by-step guide to creating a custom MapWindow Plug-in  
using the IPlugin interface

by Allen Anselmo  
*shade@turbonet.com*

## **Introduction**

The MapWindow open-source GIS platform is a system which contains a great deal of simple, straight-forward GIS functionality which overall is enough to meet the needs of many users. However, the greatest aspect of MapWindow is that it is not limited purely to the base functionality provided, but instead allows a fully extensible plug-in interface that allows users to customize their MapWindow functionality to meet their custom needs with some fairly simple .NET code.

Below is a step by step example of creating such a plug-in. In this case, the plug-in will be a simple tool to display statistics of a polyline path given a Digital Elevation Model which the path overlays.

By making use of this tutorial, you will be given the key elements for the most common plug-in interface functions and capabilities,. Further information for those topics not covered however can be found in **Appendix 1**, as well as the MapWindow documentation Wiki at <http://www.mapwindow.org/wiki>

### **Step 1:**

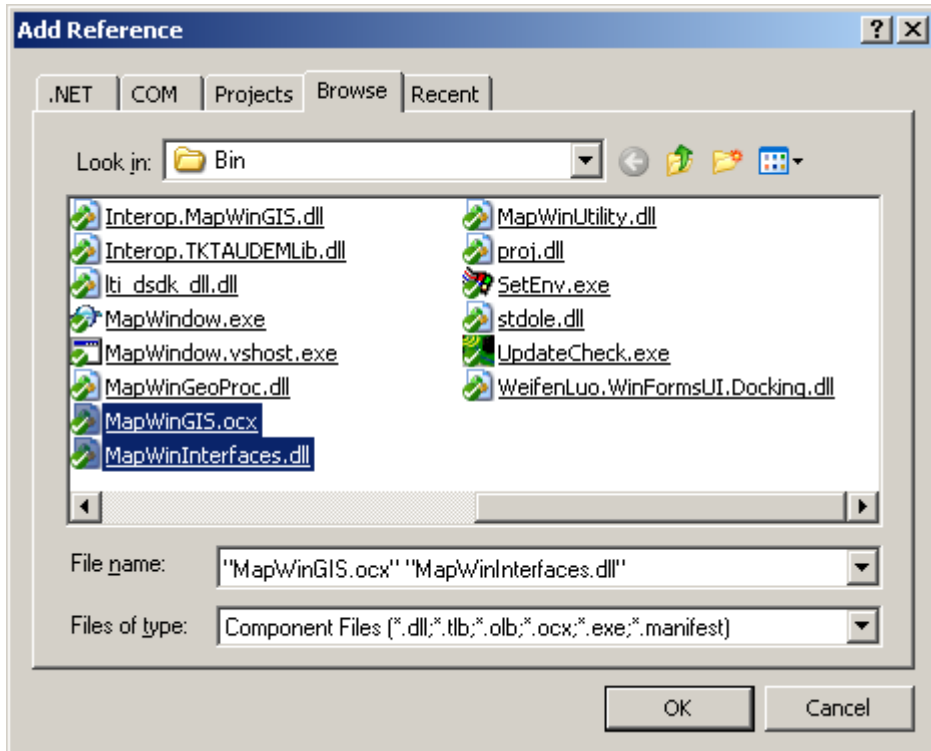
Create a new VB.Net Class Library project using Microsoft Visual Studio.

### **Step 2:**

Right-click on the class in Solution Explorer and select “Properties”

Select the References page and click the “Add” button at the bottom of that form

Then select the Browse tab and navigate to the location of and select the MapWinGIS.ocx and MapWininterfaces.dll, typically found in C:\Program Files\MapWindow\ as shown in **Figure 1** below.



**Figure 1: Adding reference to MapWinGIS and MapWinInterfaces**

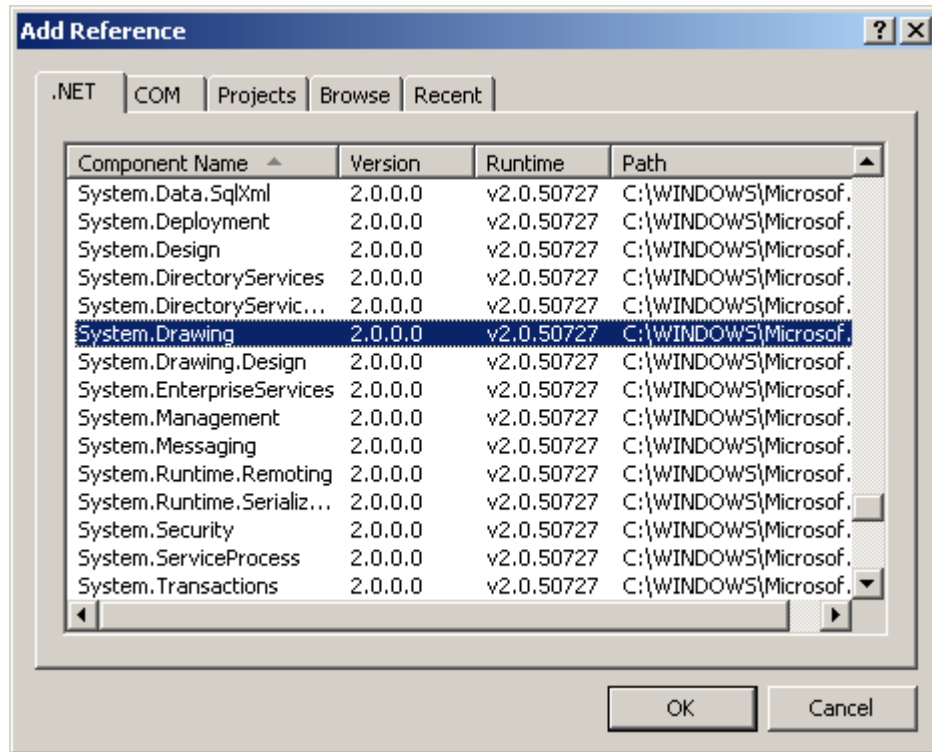
Click “OK” to add the references

Then set the Copy Local attribute of each reference to False after adding them

**Step 3:**

Click the “Add” button again and this time select the “.NET” tab

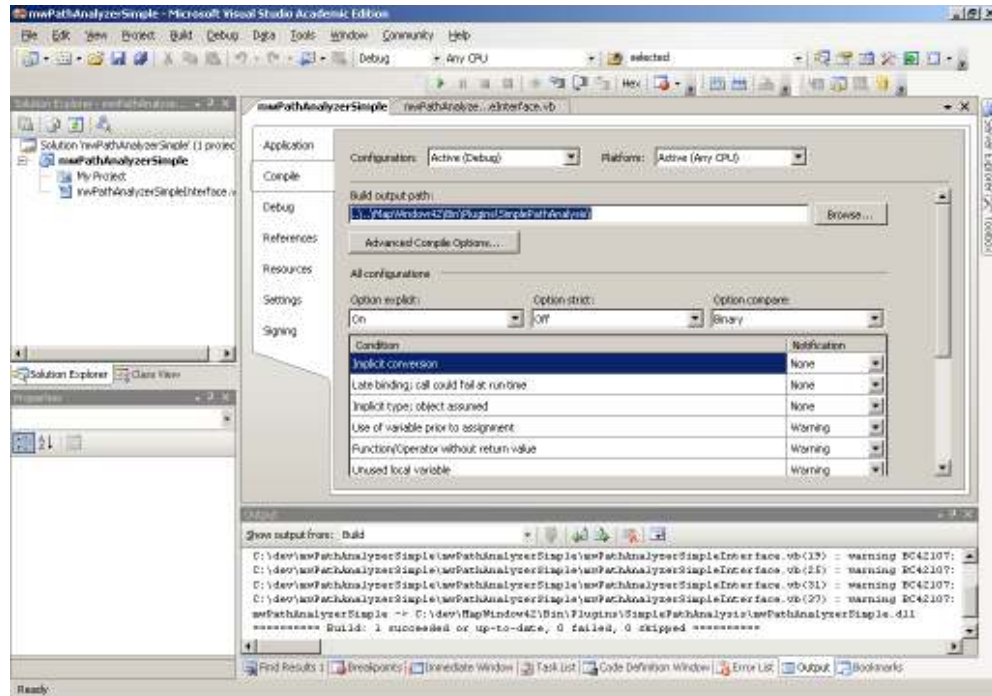
From the list of components, select “System.Drawing” and “OK”, as seen in **Figure 2**.



**Figure 2: Adding System.Drawing Reference**

### Optional Step:

In the compile settings, it is sometimes useful to set the build path to your MapWindow “Plugins” sub-directory or a sub-directory within it, such as C:\Program Files\MapWindow\Plugins\PathAnalyzer directory. As seen in **Figure 3**.



**Figure 3: Setting the Compile Build Path**

This will save you the step of having to copy your plug-in DLL into the Plugins directory for MapWindow to load when it runs.

If using this, do not forget that the path must be set for both Debug and Release properties.

### Step 4:

Back in your code, after the “Public Class...” line, insert the line **Implements MapWindow.Interfaces.IPlugin**

Hit enter on that interface and see how Visual Studio populates all of the interface properties and functions found in the IPlugin Interface.

### Step 5:

See **Appendix 1** for details on each of the properties and functions to give you a sense for how to use the plug-in interface to extend MapWindow.

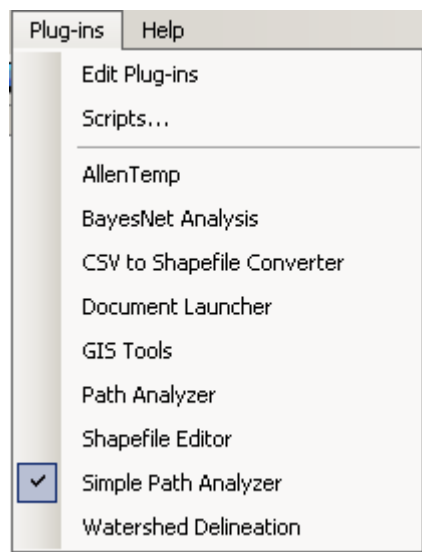
For now though, let’s start with the most important pieces.

The single most important property is “Name”. Without this returning a valid string, the plug-in will not even be loaded into the plug-in list.

So first, find the Name property and within the Get/End Get pair, type **return “Simple Path Analyzer”**

Now build the project with CTRL+SHIFT+B (and if you didn’t do the optional step to set build path, copy the resulting plug-in DLL into the Plugins directory).

Then load MapWindow, select the Plug-ins menu and see how Simple Path Analyzer is listed there now and can be turned on and off by selecting it, as seen in **Figure 4**.



**Figure 4: Simple Path Analyzer in Plug-ins Menu**

**Step 6:**

Of course, at this point, turning it on or off does nothing because its behavior on load and unload must be set through the Initialize and Terminate functions.

Most commonly, these are used to create and remove menu items and buttons from the main MapWindow interface, which we’ll do in the next step.

It is simple to do this, though it should always be done in pairs so that lingering menu items aren’t created or ones not created aren’t trying to be destroyed.

Before the properties and functions generated in **Step 4**, insert the text **private g\_MapWin as MapWindow.Interfaces.IMapWin**

Then, in the “Initialize” function, type **g\_MapWin = MapWin**

This is setting a class object to hold a reference to the main MapWindow application. When each plug-in is initialized, it's "Initialize" function triggers and is passed the MapWin object that loaded it.

This object has access to various useful objects useful for customizing the MapWindow interface, including access to menu and toolbar items, layers loaded, plug-ins loaded, status/progress bar messages, preview map capabilities, and application info.

### Step 7:

The most common use of the g\_MapWin object is to add and remove menu and toolbar items. First we'll look at menus, using the code below, which should be copied into the respective sections of your plug-in.

#### In Initialize:

```
Dim nil As Object
nil = Nothing
'Create base menu item by passing no parent menu
g_MapWin.Menus.AddMenu("btspSimplePathAnalyzer", nil,
"Watershed Delineation")
'Create sub menu items by passing parent menu of unique
identifier of base menu
g_MapWin.Menus.AddMenu("btspLoadLayers",
"btspSimplePathAnalyzer", nil, "Load Layers")
g_MapWin.Menus.AddMenu("btspAnalyze",
"btspSimplePathAnalyzer", nil, "Analyze")
```

#### In Terminate:

```
'Remove menus by unique identifiers
g_MapWin.Menus.Remove("btspLoadLayers")
g_MapWin.Menus.Remove("btspAnalyze")
g_MapWin.Menus.Remove("btspSimplePathAnalyzer")
```

Rebuild (copy the DLL if needed) and rerun MapWindow to see what happens now when you run or terminate the Simple Path Analyzer plug-in. What happens if you comment out the g\_MapWin.Menus.Remove("btspSimplePathAnalyzer") line when you run MapWindow and run or terminate the plug-in? This lingering menu issue is why you always do the creation and removal of menu and toolbar items in pairs.

### Step 8:

Next, look to how one creates a toolbar strip and icons with the following code to be copied into yours. Be sure to note that the icon file has to be added as an existing item to the solution and its type changed to embedded resource.

#### In Initialize:

```
'Add icons by creating an icon from a bitmap object and
adding to the mapwin toolbar
'Icon Analyze_16.ico has to be loaded to the solution as an
embedded resource for it to create a bitmap
Dim bt As New System.Drawing.Bitmap(Me.GetType(),
"analyze_16.ico")
```

```

Dim ico As System.Drawing.Icon =
System.Drawing.Icon.FromHandle(bt.GetHIcon)
Dim tlbButton As MapWindow.Interfaces.ToolBarButton =
MapWin.Toolbar.AddButton("sPathInfo", "sPathAnalyzer", "",
"")
tlbButton.Tooltip = "Simple Path Analyzer Tool"
tlbButton.Picture = ico

```

### In Terminate:

```

'Remove button and then toolbar
g_MapWin.Toolbar.RemoveButton("sPathInfo")
g_MapWin.Toolbar.RemoveToolbar("sPathAnalyzer")

```

Rebuild (copy the DLL if needed) and rerun MapWindow to see what happens now when you run or terminate the Simple Path Analyzer plug-in now. Is there a toolbar strip being created with a button on it?

### Step 9:

Now that you have menus and a toolbar button for the plug-in, let's make them actually do something.

To do this, we'll use the most often used function of the plug-in interface excepting Initialize/Terminate. ItemClicked.

This function triggers whenever a MapWindow button or menu item is clicked and the way that you tell what was clicked is based upon the unique name given to that item, as seen in the code below.

```

Public Sub ItemClicked(ByVal ItemName As String, ByRef Handled As
Boolean) Implements MapWindow.Interfaces.IPlugin.ItemClicked
If ItemName = "sPathInfo" Then
MsgBox("Simple Path Analyzer Toolbar Button Clicked")
ElseIf ItemName = "btspLoadLayers" Then
MsgBox("Simple Path Analyzer Load Layers Menu Item
clicked")
ElseIf ItemName = "btspAnalyze" Then
MsgBox("Simple Path Analyzer Analyze Menu Item clicked")
End If
End Sub

```

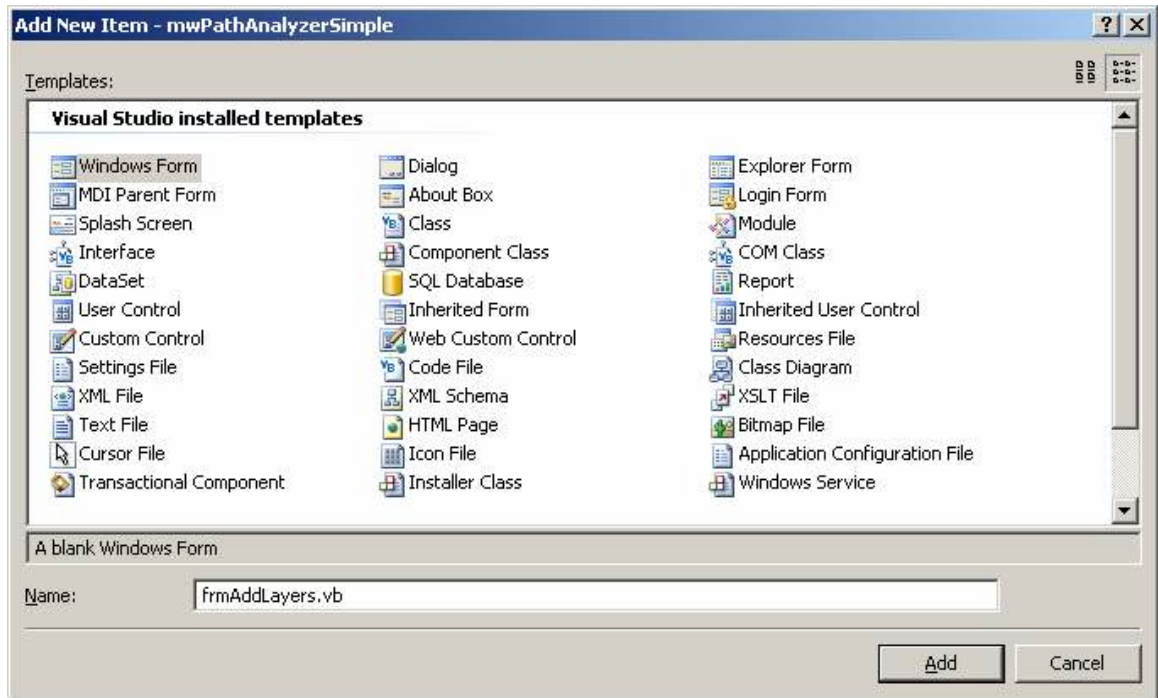
Rebuild (copy the dll to the plugins directory if needed) and rerun MapWindow to initialize the plug-in and click on each menu item and toolbar button. Is a message popping up for each correctly?

### Step 10:

Now that we've seen a basic use of the buttons, let's actually make them do something a little more interesting.

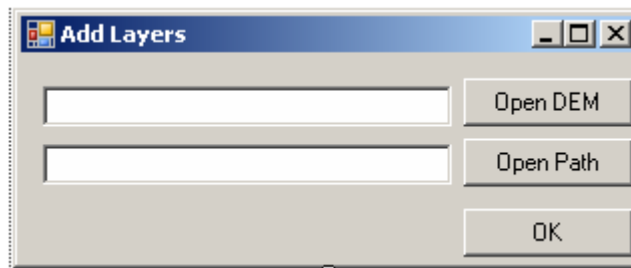
Start by creating a new Windows form in the project by right-clicking the solution in Solution Explorer and choosing the "Add" sub-menu and then "Windows Form"

Type “frmAddLayers.vb” for the name, as seen below in **Figure 5**, then click Add



**Figure 5: Adding a new form with frmAddLayers.vb name**

Once you have a new blank form, add three buttons and two textboxes to look like **Figure 6** below and change their properties as below. Also add an OpenFileDialog object to the form



**Figure 6: Add Layers form layout**

Form:

Text to “Add Layers”

Bottom Button:

Text to “OK”

Name to “btnOK”

DialogResult to “OK”

Top Button:

Text to "Open DEM"  
Name to "btnOpenDEM"

Middle Button:  
Text to "Open Path"  
Name to "btnOpenPath"

Top Textbox:  
Name to "txtbxDEM"

Bottom Textbox:  
Name to "txtbxPath"

OpenFileDialog:  
Name to "fdiagOpen"

### Step 11:

Right click on the form and select "View Code" then insert the following code for handling the form.

```
Public DEMpath As String
Public Pathpath As String

Private Sub btnOpenDEM_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOpenDEM.Click
    Dim g As New MapWinGIS.Grid
    fdiagOpen.Filter = g.CdlgFilter

    If fdiagOpen.ShowDialog() Then
        txtbxDEM.Text = fdiagOpen.FileName
        DEMpath = txtbxDEM.Text
    End If
End Sub

Private Sub btnOpenPath_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOpenPath.Click
    Dim sf As New MapWinGIS.Shapefile
    fdiagOpen.Filter = sf.CdlgFilter

    If fdiagOpen.ShowDialog() Then
        'Open Shapefile to test it for being a polyline type
        If sf.Open(fdiagOpen.FileName) Then
            If sf.ShapefileType = MapWinGIS.ShpfileType.SHP_POLYLINE Then
                txtbxPath.Text = fdiagOpen.FileName
                Pathpath = txtbxPath.Text
            Else
                MsgBox("The path shapefile must be a polyline shapefile")
            End If
        End If
    End If
End Sub
```

```

        'close shapefile object
        sf.Close()
    End If
End If
End Sub

Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnOK.Click
    If txtbxPath.Text = "" Or txtbxDEM.Text = "" Or Not
System.IO.File.Exists(txtbxPath.Text) Or Not
System.IO.File.Exists(txtbxDEM.Text) Then
        MsgBox("You must add a filepath for both the DEM and
Path values.")
        Me.DialogResult = Windows.Forms.DialogResult.None
    End If
End Sub

```

Now return to the main plug-in code and add the following Sub and class variables

```

Private DEMpath As String
Private Pathpath As String

Private Sub LoadLayers()
    g_MapWin.Layers.Clear()
    DEMpath = ""
    Pathpath = ""
    Dim frmLoad As New frmAddLayers
    If frmLoad.ShowDialog = Windows.Forms.DialogResult.OK Then
        'Set the class variables for paths to be used
        DEMpath = frmLoad.DEMpath
        Pathpath = frmLoad.Pathpath
        'Adds the pathes as layers to the map
        g_MapWin.Layers.Add(DEMpath)
        g_MapWin.Layers.Add(Pathpath)
    End If
End Sub

```

### Step 12:

Then in the “ItemClicked” function modify it to the following to load layers on appropriate clicks and specifically pay attention to how you are able to set the cursor mode via the `g_MapWin.View.CursorMode` in order to set it into selection mode.

```

Public Sub ItemClicked(ByVal ItemName As String, ByRef Handled As
Boolean) Implements MapWindow.Interfaces.IPlugin.ItemClicked
    If ItemName = "sPathInfo" Then
        'Only prompt if they haven't already set the pathes
        If DEMpath = "" And Pathpath = "" Then
            LoadLayers()
        End If
        'Set into selection mode so when they select a path
and trigger ShapesSelected
        g_MapWin.View.CursorMode =
MapWinGIS.tkCursorMode.cmSelection
    End If
End Sub

```

```

ElseIf ItemName = "btspLoadLayers" Then
    LoadLayers()
ElseIf ItemName = "btspAnalyze" Then
    'Set into selection mode so when they select a path
    and trigger ShapesSelected
    g_MapWin.View.CursorMode =
    MapWinGIS.tkCursorMode.cmSelection
End If
End Sub

```

Now rebuild (copy the DLL if needed) and rerun MapWindow to click on the menu and toolbar button to see how it adds layers and sets into select mode.

### Step 13:

Now that we've got the buttons loading the needed layers and setting into a seemingly pointless selection mode, how do we actually make use of it?

The answer is to use another plug-in interface function which is often useful in many types of plug-ins. ShapesSelected.

This function will trigger whenever shapes are selected from a shapefile layer in the map. In this case, we will use this to calculate some simple statistics of the path selected.

To do this, insert the following code in to the ShapesSelected interface function

```

Dim sf As MapWinGIS.Shapefile
Dim selectedShapeIdx As Int16

If DEMpath <> "" And Pathpath <> "" Then
    'make sure something was returned in SelectInfo
    If SelectInfo Is Nothing Then Exit Sub
    'Grab the index of the first of selected shapes (in case
    multiple shapes selected)
    selectedShapeIdx = SelectInfo(0).ShapeIndex
    'Get the shapefile associated with current layer handle
    being selected from
    sf = g_MapWin.Layers(Handle).GetObject
    ShowPathStats(sf.Shape(selectedShapeIdx))
End If

```

Then add the private sub below:

```

Private Sub ShowPathStats(ByVal path As MapWinGIS.Shape)
    Dim currElev, minElev, maxElev, avgElev, totElev,
    currLength, totLength As Double
    Dim currCol, currRow As Integer
    Dim g As New MapWinGIS.Grid
    Dim currPt1, currPt2 As MapWinGIS.Point
    minElev = 10000
    maxElev = -10000
    avgElev = 0
    totElev = 0

```

```

totLength = 0

'Open the DEM grid
g.Open(DEMpath)

'cycle the vertex points of the shape passed
For i As Integer = 0 To path.numPoints - 1
    currPt1 = path.Point(i)
    'Find the grid cell associated with each point
    g.ProjToCell(currPt1.x, currPt1.y, currCol, currRow)
    'Get elevation at that point
    currElev = g.Value(currCol, currRow)
    'Check for min/max elevations
    If currElev > maxElev Then
        maxElev = currElev
    End If
    If currElev < minElev Then
        minElev = currElev
    End If
    totElev = totElev + currElev
Next
avgElev = totElev / path.numPoints
g.Close()

'cycle from 1 up to get lengths between each vertex
For i As Integer = 1 To path.numPoints - 1
    currPt1 = path.Point(i - 1)
    currPt2 = path.Point(i)

    currLength = Math.Sqrt(Math.Pow(currPt2.x -
    currPt1.x, 2) + Math.Pow(currPt2.y - currPt1.y, 2))
    totLength = totLength + currLength
Next

'Show a message with the data
MsgBox("Min Elevation: " + minElev.ToString + vbNewLine +
"Max Elevation: " + maxElev.ToString + vbNewLine + "Average
Elevation: " + avgElev.ToString + vbNewLine + "Total
Length: " + totLength.ToString)
End Sub

```

#### Step 14:

With that done, rebuild (copy the DLL if needed) and rerun MapWindow and click the toolbar button, select a DEM and polyline shapefile, click OK, then select a path and watch it display the simple stats for that path.

Congratulations! You've created your first MapWindow plug-in. Hopefully you can see how useful

# Appendix 1: Plug-in Framework

- Public ReadOnly Property Name() As String Implements MapWindow.Interfaces.IPlugin.Name
  - Name of the plugin as it will show on the Plugin menu
  - If name not specified, plugin won't load
- Public ReadOnly Property Author() As String Implements MapWindow.Interfaces.IPlugin.Author
  - Author name
- Public ReadOnly Property Description() As String Implements MapWindow.Interfaces.IPlugin.Description
  - Description of plugin
- Public ReadOnly Property BuildDate() As String Implements MapWindow.Interfaces.IPlugin.BuildDate
  - Often uses System.IO.File.GetLastWriteTime(Me.GetType().Assembly.Location)
- Public ReadOnly Property Version() As String Implements MapWindow.Interfaces.IPlugin.Version
  - Often uses System.Diagnostics.FileVersionInfo.GetVersionInfo(Me.GetType().Assembly.Location).FileVersion

## Subs to Implement

- On all functions, setting Handled to true will stop it from being sent to other plugins to enact
- Public Sub Initialize(ByVal MapWin As MapWindow.Interfaces.IMapWin, ByVal ParentHandle As Integer) Implements MapWindow.Interfaces.IPlugin.Initialize
  - Activates when plugin loads at mapwindow startup or activation from plugin menu
  - Standard to set a global g\_MapWin = MapWin for access to mapwindow data
- Public Sub Terminate() Implements MapWindow.Interfaces.IPlugin.Terminate
  - Activates on unload of plugin or closing of mapwindow
  - Used mostly for removing buttons used by plugin
- Public Sub ItemClicked(ByVal ItemName As String, ByRef Handled As Boolean) Implements MapWindow.Interfaces.IPlugin.ItemClicked

- Triggers when a button or menu item is clicked. Can tell by internal name of button whether to activate code
- Public Sub LayerRemoved(ByVal Handle As Integer) Implements MapWindow.Interfaces.IPlugin.LayerRemoved
  - Fires when a layer is removed, in case plugin relies on specific layer
- Public Sub LayersAdded(ByVal Layers() As MapWindow.Interfaces.Layer) Implements MapWindow.Interfaces.IPlugin.LayersAdded
  - Fires when new layer is added, in case plugin relies on specific layers or maintains its own internal list of certain layers
- Public Sub LayersCleared() Implements MapWindow.Interfaces.IPlugin.LayersCleared
  - Fires when user clears all layers, useful for above reasons
- Public Sub LayerSelected(ByVal Handle As Integer) Implements MapWindow.Interfaces.IPlugin.LayerSelected
  - Fired when user selects layer
- Public Sub LegendDoubleClick(ByVal Handle As Integer, ByVal Location As MapWindow.Interfaces.ClickLocation, ByRef Handled As Boolean) Implements MapWindow.Interfaces.IPlugin.LegendDoubleClick
  - Fired when user double-clicks on a layer in the legend box, which currently will open up the layer properties, but can have further functionality
- Public Sub LegendMouseDown(ByVal Handle As Integer, ByVal Button As Integer, ByVal Location As MapWindow.Interfaces.ClickLocation, ByRef Handled As Boolean) Implements MapWindow.Interfaces.IPlugin.LegendMouseDown
  - Fires on normal mouseDown event in the legend box
- Public Sub LegendMouseUp(ByVal Handle As Integer, ByVal Button As Integer, ByVal Location As MapWindow.Interfaces.ClickLocation, ByRef Handled As Boolean) Implements MapWindow.Interfaces.IPlugin.LegendMouseUp
  - Fires on normal mouseUp event in the legend box
- Public Sub MapDragFinished(ByVal Bounds As System.Drawing.Rectangle, ByRef Handled As Boolean) Implements MapWindow.Interfaces.IPlugin.MapDragFinished
  - Fires after the user has finished a drag box on the map component, and gives the bounds as a drawing rectangle
- Public Sub MapExtentsChanged() Implements MapWindow.Interfaces.IPlugin.MapExtentsChanged

- Fires when user zooms or pans to change map extents
- Public Sub MapMouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Integer, ByVal y As Integer, ByRef Handled As Boolean)  
Implements MapWindow.Interfaces.IPlugin.MapMouseDown
  - Fires on click of the map component and tells which button, shift state, and point clicked on
  - X and Y are in screen coordinates, not map, so if you need map coordinates, must use g\_MapWin.View.PixelToProj()
- Public Sub MapMouseMove(ByVal ScreenX As Integer, ByVal ScreenY As Integer, ByRef Handled As Boolean) Implements MapWindow.Interfaces.IPlugin.MapMouseMove
  - Fires on mouse move across the map component
  - Coordinates again in screen format, as above
- Public Sub MapMouseUp(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Integer, ByVal y As Integer, ByRef Handled As Boolean)  
Implements MapWindow.Interfaces.IPlugin.MapMouseUp
  - Fires on mouse up on the map component
  - Coordinates still in screen form
- Public Sub Message(ByVal msg As String, ByRef Handled As Boolean)  
Implements MapWindow.Interfaces.IPlugin.Message
  - Fired when a plugin sends out a message which will be redirected to all other plugins to allow communication
  - If a plugin determines message is “For” them, they can set handled to true to stop it sending to others
- Public Sub ProjectLoading(ByVal ProjectFile As String, ByVal SettingsString As String) Implements MapWindow.Interfaces.IPlugin.ProjectLoading
  - Fires when user opens a project file
  - SettingString holds information saved by the plugin at the project level, such as a database path associated with the project specifically
- Public Sub ProjectSaving(ByVal ProjectFile As String, ByRef SettingsString As String) Implements MapWindow.Interfaces.IPlugin.ProjectSaving
  - Fires when user saves a project
  - SettingString allows the plugin to save information tied to plugin at the project level, such as a database path associated with that project specifically
- Public Sub ShapesSelected(ByVal Handle As Integer, ByVal SelectInfo As MapWindow.Interfaces.SelectInfo) Implements MapWindow.Interfaces.IPlugin.ShapesSelected
  - Fires when shapes are selected in a shapefile layer

- SelectInfo holds which were selected