By Ben Sitler

Introduction

First of all, if you are looking to write your own Grasshopper components you are likely 1) an experienced Grasshopper user and 2) quickly outgrowing the VB scripting components. You are looking for more functionality and a more permanent fix than transient scripted components in your .ghx files. Fortunately for you, writing our own custom grasshopper components can be but a relatively simple and pain free next step from scripting if you follow these instructions.

Grasshopper is a DotNET plugin for Rhino, and any addition to Grasshopper must be written with VB.NET or C#.NET (Actually any DotNET language would work and even C++ or CLI would be technically feasible, but these alternative routes would be a BIG headache). David uses both C#.NET and VB.NET, and I will show you how to view existing components in either language to learn the syntax and methods of implementation. It is only a matter of preference between the two as to which one you use. To get started you are going to need to obtain some software:

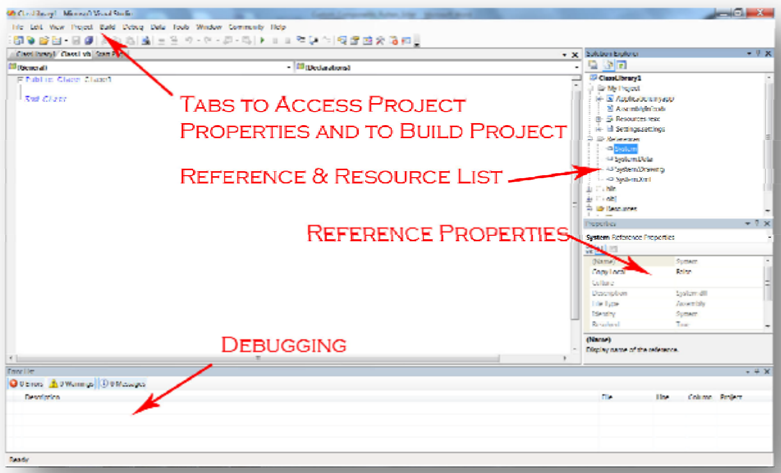
- Visual Studio (express editions free at http://msdn.microsoft.com/en-us/vstudio/default.aspx, VS Standard 2005 can be bought on Amazon or a similar place for less than $100)
- Rhino DotNET SDK (http://download.rhino3d.com/Rhino/3.0/DotNetSDK/)
- Redgate's .NET Reflector (free at http://www.red-gate.com/products/reflector/index.htm)
- Rhino and Grasshopper, obviously

Note: This manual is based on VB.NET using Visual Studio 2005 Standard Edition (what I use!), but it is easy to apply these steps to VS 2008 or 2010, Express Editions and C#.NET.

Once everything is downloaded and installed follow the instructions in the Rhino DotNET SDK "readme" file.

An important note is that when Rhino 5.0 ships, **NONE OF YOUR COMPONENTS WILL BE COMPATIBILE**. The new version will ship with a new DotNET SDK called RhinoCommon. While there does not seem to be a precise date, realize that Grasshopper 0.7.X (built on the new SDK) could become available as soon as summer 2010. Therefore, any custom component development that you do in the next few months should only be to get your feet wet, not to program any serious plugins.

Alas, now that we have washed our hands of the upcoming tragedy we can refocus on learning the Visual Studio environment and creating custom Grasshopper components.


Tabs to Access Project Properties and to Build Project
Reference & Resource List
Reference Properties
Debugging

Setting up a class library

Open up Visual Studio and create a New Project of type Class Library (Chose a project type of VB or C# at this stage depending on your preference…you cannot change this latter). Enter in the project name.

Next you need to create the references. Click Toolbar: Project>Add Reference, and then add each of the following, one at a time (set CopyLocal = False in reference properties window...see above image):

- Rhino_DotNET.dll (Browse>found in the Rhino System folder, next to Rhino.exe)
- Grasshopper.dll (Browse>found in the Grasshopper plug-in folder)
- GH_IO.dll (Browse>found next to Grasshopper.dll)
- System.Drawing (.NET)
- System.Windows.Forms (.NET)

If you have VS Pro or Standard edition, go to Toolbar: Project>*Name* Properties>Compile>Build Events and in the Post Build Event Command Line box enter the following code>OK

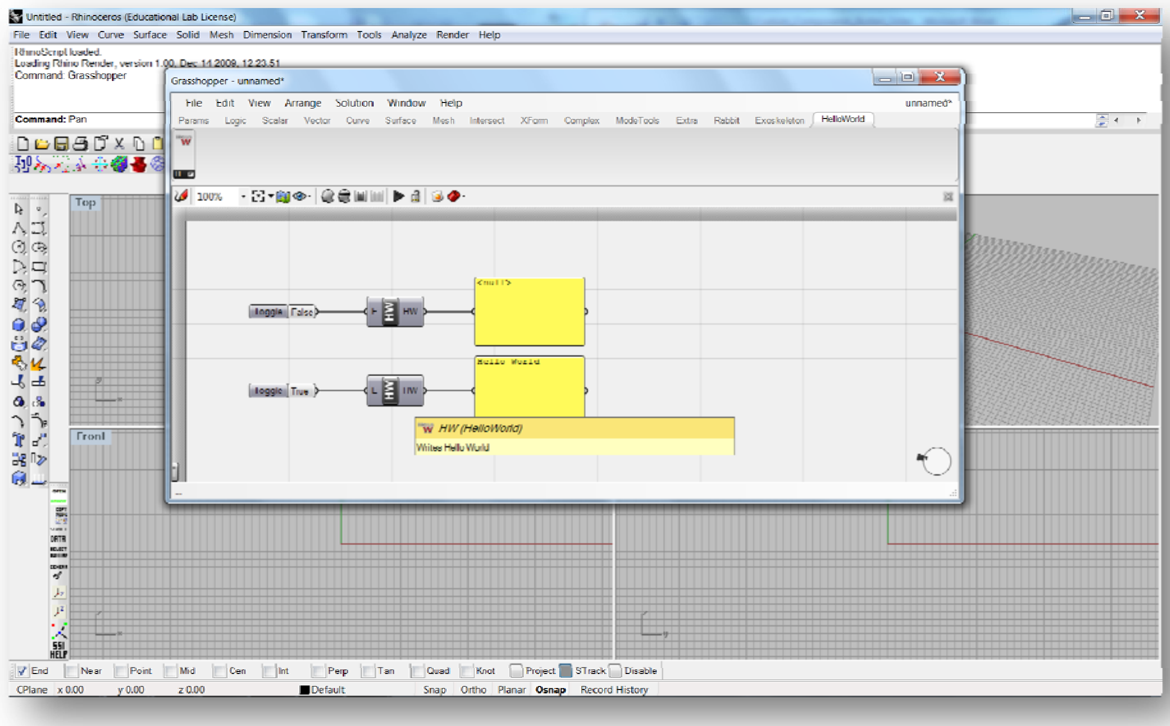
rename "$(Targetpath)" "*Name*.gha"

If you have VS Express you will have rename the plugin .dll extension to .gha after each build and move the program manually to C:\Program Files\Rhinoceros 4.0\Plug-ins\Grasshopper\Components

(the .dll can be found in Documents>Visual Studio>Projects>*Name*>Bin>Release>*Name*.dll)

Creating your first Class (Grasshopper buttons)

In the tradition of programming, our first component will be “Hello World.” Copy and paste the following VB or C# code into the Visual Studio editor. Your program will look like this:


```
RhinoScript loaded.
Loading Rhino Render, version 1.00, Dec 14 2008, 12:23:51
Command: Grasshopper
Command: Pan
```

HelloWorld in Visual Basic.NET

```
Imports Grasshopper.Kernel
Imports Grasshopper.Kernel.Types
Imports RMA.OpenNURBS
Public Class HelloWorldVB
    Inherits Grasshopper.Kernel.GH_Component
    ' Methods
    Public Sub New()
        MyBase.New("HelloWorld", "HW", "Writes Hello World", "Extra", "HelloWorld")
    End Sub
' This is "ButtonName", "Abbreviation", "Description", "TabName", "SubTabName"
' The TabName can be either any of the existing ribbon tab names or a new one of your
'   creation.  The same goes for the SubTabName and if it matches an existing TabName,
'   your button will simply be added to this tab.
    Protected Overrides Sub RegisterInputParams(ByVal pManager As GH_InputParamManager)
        pManager.Register_Boolean("Execute", "E", "Execute?", False, False)
    End Sub
' InputParams can be anything from integers, on3dpoint, booleans or any other valid
'   parameter listed by the tooltip that appears when pManager. is typed.  If your input
'   is not a list, make sure the last boolean is set to False
    Protected Overrides Sub RegisterOutputParams(ByVal pManager As GH_OutputParamManager)
        pManager.Register_StringParam("String", "S", "HelloWorld String")
    End Sub
    Protected Overrides Sub SolveInstance(ByVal Da As IGH_DataAccess)
        Dim Execute As Boolean = False
        If (Da.GetData(Of GH_Boolean)(0, Execute)) Then
            If Execute Then
                Da.SetData(0, "Hello World")
            End If
        End If
    End Sub
' Your InputParams are accessed using DA.GetData(Of type)(index, variable stored to) from
'   indexes 0 to nInputs-1 and you outputs are assigned using iData.SetData(index,value)
'   with indexes 0 to nOutputs-1.  It is good practice to use an If Statement to prevent
'   acting with empty variables.
    ' Properties
    Public Overrides ReadOnly Property ComponentGuid() As Guid
        Get
            Return New Guid("{3cc52f82-c4bb-4520-85b4-34f3b826658b}")
        End Get
    End Property
' You must create your own unique Guid, see below
    Protected Overrides ReadOnly Property Internal_Icon_24x24() As System.Drawing.Bitmap
        Get
            Return My.Resources.HelloWorld
        End Get
    End Property
' You must create your own image first before you can use the resource.
End Class
```

HelloWorld in C#.NET

```
using System; using System.Collections.Generic; using System.Text; using System.Drawing;
using Grasshopper.Kernel; using Grasshopper.Kernel.Types; using RMA.OpenNURBS;
namespace HelloWorldC {
    public class HelloWorld : GH_Component {
        // Methods
        public HelloWorld()
            : base("HelloWorld", "HW", "Writes Hello World", "Extra", "HelloWorld"){
        }
        protected override void RegisterInputParams(GH_Component.GH_InputParamManager
pManager){
            pManager.Register_BooleanParam("Execute", "E", "Execute?", false, false);
        }
        protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager
pManager){
            pManager.Register_StringParam("HelloWorld", "HW", "HelloWorld String");
        }
        protected override void SolveInstance(IGH_DataAccess DA){
            GH_Boolean Execute = null;
            if (DA.GetData<GH_Boolean>(0, out Execute)){
                if (Execute){
                    DA.SetData(0, "Hello World");
                }
            }
        }
        // Properties
        public override Guid ComponentGuid {
            get {
                return new Guid("{c68f4c6b-93d7-4e8d-bf83-1bdcad2ee37c}");
            }
        }
        protected override Bitmap Internal_Icon_24x24 {
            get {
                return HelloWorldC.Properties.Resources.HelloWorld;
            }
        }
    }
}
```

Note: To make this program work, you need to create a few more things.

Guid stands for Globally Unique Identifier and is a pseudo unique ID number for your program. You can generate these at http://www.guidgenerator.com/online-guid-generator.aspx and you need a new GUID for each class that you create.

To create your button image you have essentially two options: you can make a quickie 24x24 pixel bitmap using the built in editor, or you can import an image created in a stand alone editor like Photoshop or XaraX. The key to a good quality image is a transparent background (.PNG are great for this purpose) and anti aliased object borders (to avoid pixilation). Valid image types are .PGN, .BMP, .GIF, .JPG, .TIFF.

VS Editor: Toolbar: Project>*Name* Properties>Resources>Add Resource>New Image>BMP>Enter Name

Change the image size to 24x24 pixels and draw away! Then go back and `Return My.Resources.ImageName` to the name of your image.

External Editor: Same except Add Resource>Add Existing File>locate image filepath

Now you are ready to debug. Toolbar: Build>Build *Name* (VS Express you will now need to manually change the extension as mentioned previously) and Copy/Paste .gha file into Plugins>Grasshopper>Components

Open Grasshopper and if you were successful (and remembered to create a new GUID) your component will load in its appropriate place on the Grasshopper ribbon. Notice that you can call your component in all the usual ways, including via double click and typing in its name. If there is a logic error go back to Visual Studio and debug.

Disassembling Existing Grasshopper Components

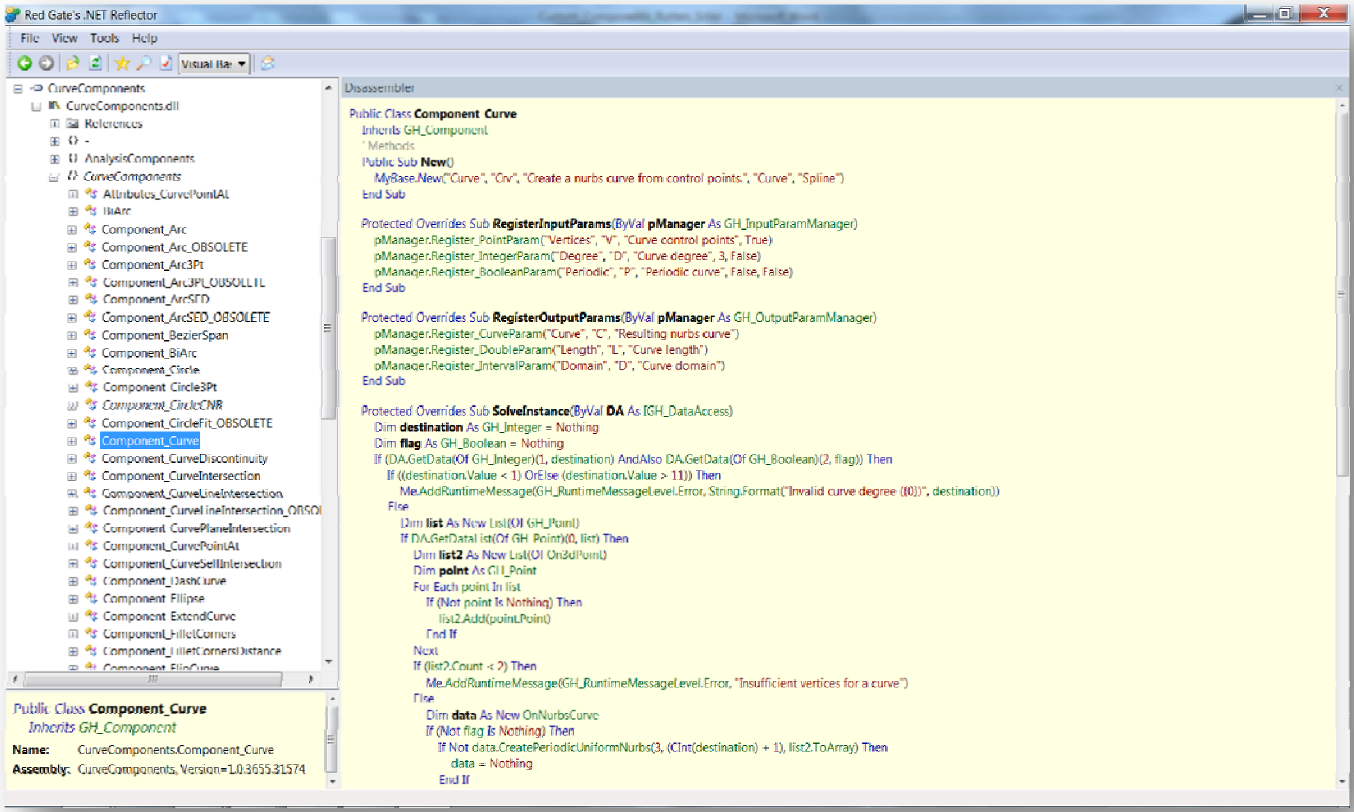
At this point you may be wondering what the deal was with downloading Reflector. Well here's why: Reflector allows you to "disassemble" existing executables and dll's to see what the program is actually doing behind the scenes. I HIGHLY RECOMMEND for you to go File>Open in Reflector for Grasshopper.dll, and each of the existing component .gha files. An example of how to disassemble files follows:

File>Open> C:\Program Files\Rhinoceros 4.0\Plug-ins\Grasshopper\Components\Curve (select view all files)

Menu Window>CurveComponents>CurveComponents.dll> CurveComponents. ComponentCurve

At this point you can set the DotNET language that you would like to disassemble the program with.

Then with the component highlighted, Tools>Disassemble or click Expand Methods



```
CurveComponents
  CurveComponents.dll
    References
    {} -
    {} AnalysisComponents
    {} CurveComponents
      Attributes_CurvePointAt
      BiArc
      Component_Arc
      Component_Arc_OBSOLETE
      Component_Arc3Pt
      Component_Arc3Pt_OBSOLETE
      Component_ArcSED
      Component_ArcSED_OBSOLETE
      Component_BezierSpan
      Component_BiArc
      Component_Circle
      Component_Circle3Pt
      Component_CircleCNR
      Component_CircleFit_OBSOLETE
      Component_Curve
      Component_CurveDiscontinuity
      Component_CurveIntersection
      Component_CurveLineIntersection
      Component_CurveLineIntersection_OBSOL
      Component_CurvePlaneIntersection
      Component_CurvePointAt
      Component_CurveSelfIntersection
      Component_DashCurve
      Component_Ellipse
      Component_ExtendCurve
      Component_FilletCorners
      Component_FilletCornersDistance
      Component_FlipCurve
```

```
Public Class Component_Curve
    Inherits GH_Component
Name:      CurveComponents.Component_Curve
Assembly:  CurveComponents, Version=1.0.3655.31574
```

```
Public Class Component_Curve
    Inherits GH_Component
    ' Methods
    Public Sub New()
        MyBase.New("Curve", "Crv", "Create a nurbs curve from control points.", "Curve", "Spline")
    End Sub

    Protected Overrides Sub RegisterInputParams(ByVal pManager As GH_InputParamManager)
        pManager.Register_PointParam("Vertices", "V", "Curve control points", True)
        pManager.Register_IntegerParam("Degree", "D", "Curve degree", 3, False)
        pManager.Register_BooleanParam("Periodic", "P", "Periodic curve", False, False)
    End Sub

    Protected Overrides Sub RegisterOutputParams(ByVal pManager As GH_OutputParamManager)
        pManager.Register_CurveParam("Curve", "C", "Resulting nurbs curve")
        pManager.Register_DoubleParam("Length", "L", "Curve length")
        pManager.Register_IntervalParam("Domain", "D", "Curve domain")
    End Sub

    Protected Overrides Sub SolveInstance(ByVal DA As IGH_DataAccess)
        Dim destination As GH_Integer = Nothing
        Dim flag As GH_Boolean = Nothing
        If (DA.GetData(Of GH_Integer)(1, destination) AndAlso DA.GetData(Of GH_Boolean)(2, flag)) Then
            If ((destination.Value < 1) OrElse (destination.Value > 11)) Then
                Me.AddRuntimeMessage(GH_RuntimeMessageLevel.Error, String.Format("Invalid curve degree ({0})", destination))
            Else
                Dim list As New List(Of GH_Point)
                If DA.GetDataList(Of GH_Point)(0, list) Then
                    Dim list2 As New List(Of On3dPoint)
                    Dim point As GH_Point
                    For Each point In list
                        If (Not point Is Nothing) Then
                            list2.Add(point.Point)
                        End If
                    Next
                    If (list2.Count < 2) Then
                        Me.AddRuntimeMessage(GH_RuntimeMessageLevel.Error, "Insufficient vertices for a curve")
                    Else
                        Dim data As New OnNurbsCurve
                        If (Not flag Is Nothing) Then
                            If Not data.CreatePeriodicUniformNurbs(3, (CInt(destination) + 1), list2.ToArray) Then
                                data = Nothing
                            End If
```

Additional Tips

If you want to speed up the debugging process you can adjust the .gha loading logic via GrasshopperDeveloperSettings command in Rhino.

You will likely notice once you start looking at David's components that he often includes two more overrides that are labeled HelpDescription and Exposure. The defaults of these are sufficient for the novice developer, but as your knowledge increase you want to know what these do. The **HelpDescription** override is the help file that pops up when you right click on a component and press Help. Its default is the component's Description string. **Exposure** controls where on the GUI your component icon will be visible.

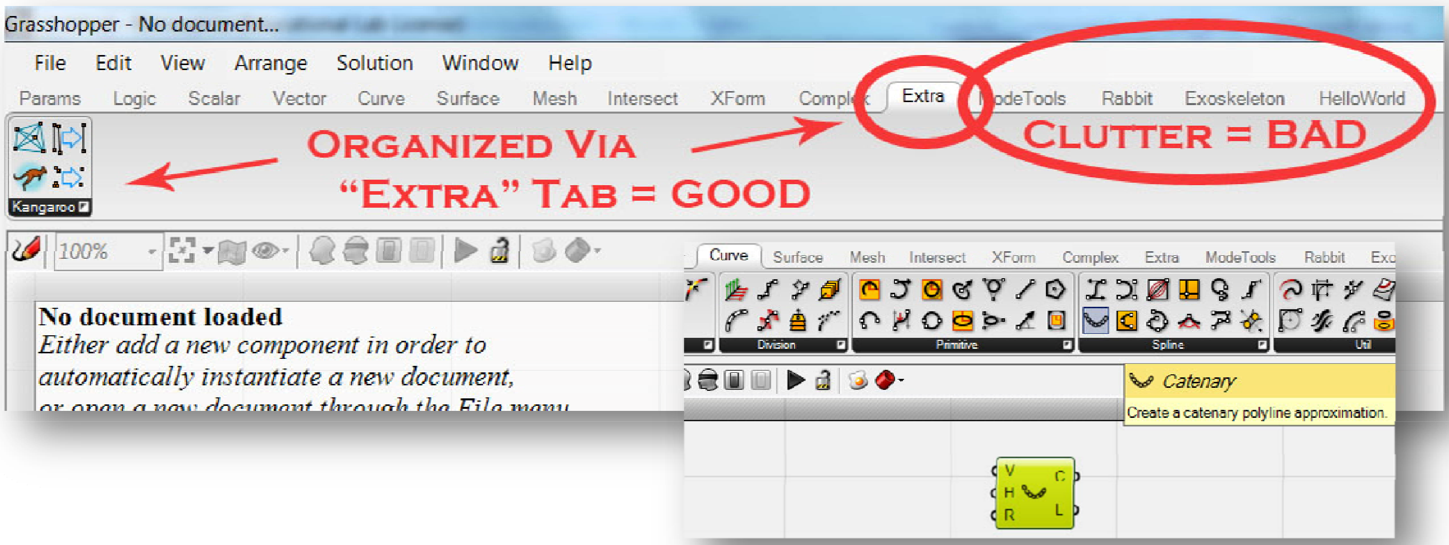
If you are using a Visual Studio Express Edition, check out http://wiki.mcneel.com/developer/dotnetplugins

For programming syntax and reference, use http://msdn.microsoft.com/en-us/library/default.aspx

When you are programming it is helpful to access the tooltips (the menu that pops up after you type a dot with all of the sub members) even after they have disappeared. The Windows shortcut key is Ctrl + Space

Guidance on New Component Ribbon Location

To avoid the inherent clutter of dozens of developers sharing individual components each with unique tab names, I propose a best practice of storing components on the "Extra" tab. The sub tab name could be specific to your plugin (ex. "Kangaroo" sub tab) and if your component matches the "Curve" tab better, then you should place it on that tab. And obviously, if you develop a plugin suite of over 60 components, then you warrant your own tab. Below is an illustration of my proposal.



No document loaded
Either add a new component in order to automatically instantiate a new document, or open a new document through the File menu

Create a catenary polyline approximation.

This manual would not have been possible without the contributions of David Rutten of Robert McNeel & Associates. Much of the contents are derived from an email dialogue from the past few months.

Ben Sitler
bsitler@princeton.edu
Princeton University
Princeton, NJ