Adroit SmartUI Scripting Reference

Integration and Extension

Product Version: 8.0

Quick Start Guide

1. Table of contents

2. Overview

The Adroit SmartUI provides a powerful and flexible platform for extending the product to provide additional functionality using the .NET framework. Various examples are provided in C#, although VB.NET can also be used.

This focus of this document is on the scripting environment provided in the Adroit SmartUI. For product integration and extension please see: Adroit Product Extension and Integration guide.

3. Purpose of this guide

This guide is intended to provide a user with enough information to understand the scripting environment of the Adroit SmartUI. This document is written to users already armed with a good understanding of the basic concepts within the product as well as some programming knowledge.

This guide should be used in conjunction with the Adroit Product Help Documentation (included with the product) and the Adroit Knowledge Base: http://adroitkb2.cloudapp.net

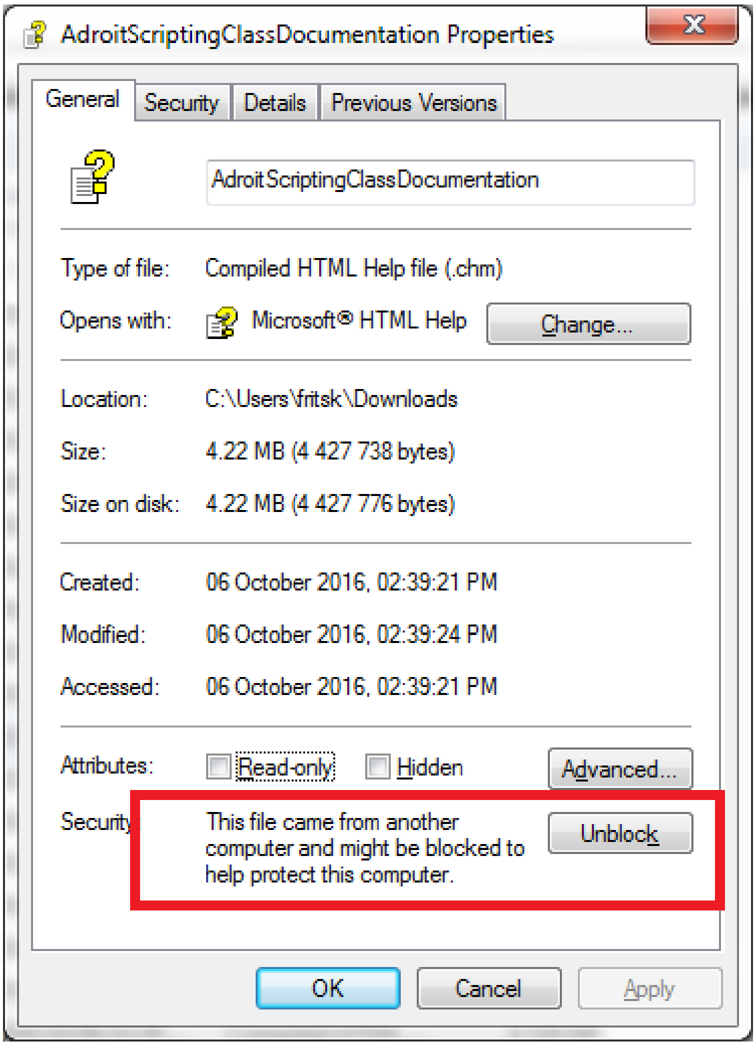
Please note that scripting is an advanced feature of the product. Though scripting offers power and flexibility, it needs to be used responsibly. Untested or careless scripting can cause product instability. Adroit accepts no liability or responsibility for the incorrect and/or misuse of the scripting functionality within the product.

4. The Libraries

The following DLLs are normally required as references in your script:

1. VIZNET.UI.SHARED.DLL
   a. Common classes shared between the Operator and Designer interfaces
2. VIZNET.UI.RUNTIME.DLL
   a. Classes relevant to the RUNTIME user interface i.e. the Operator and Designer in "Run" mode
3. VIZNET.UI.CONTROLS.DLL
   a. User interface controls
4. VIZNET.SHARED.DLL
   a. Various core classes that are shared between all components
5. Class documentation for the script classes can be downloaded here: http://adroitdemo2.cloudapp.net/KBArticles/AdroitScriptingClassDocumentation.chm

   If you cannot read the contents of the Class Documentation, it could be because it is blocked by Windows. You can unblock it in the File Properties in Windows Explorer:



AdroitScriptingClassDocumentation Properties

AdroitScriptingClassDocumentation

Type of file: Compiled HTML Help file (.chm)

Opens with: Microsoft® HTML Help

Location: C:\Users\fritsk\Downloads

Size: 4.22 MB (4 427 738 bytes)

Size on disk: 4.22 MB (4 427 776 bytes)

Created: 06 October 2016, 02:39:21 PM

Modified: 06 October 2016, 02:39:24 PM

Accessed: 06 October 2016, 02:39:21 PM

Attributes: □ Read-only □ Hidden

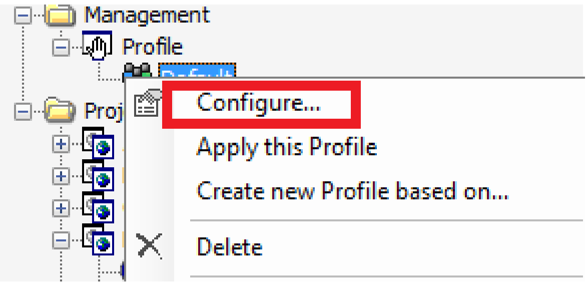
Security: This file came from another computer and might be blocked to help protect this computer.

[Sidenote: The VIZNET nomenclature was the codename for the SmartUI project that was initiated in 2001 on the .NET Framework 1.0. It stayed on as the root namespace for all classes]

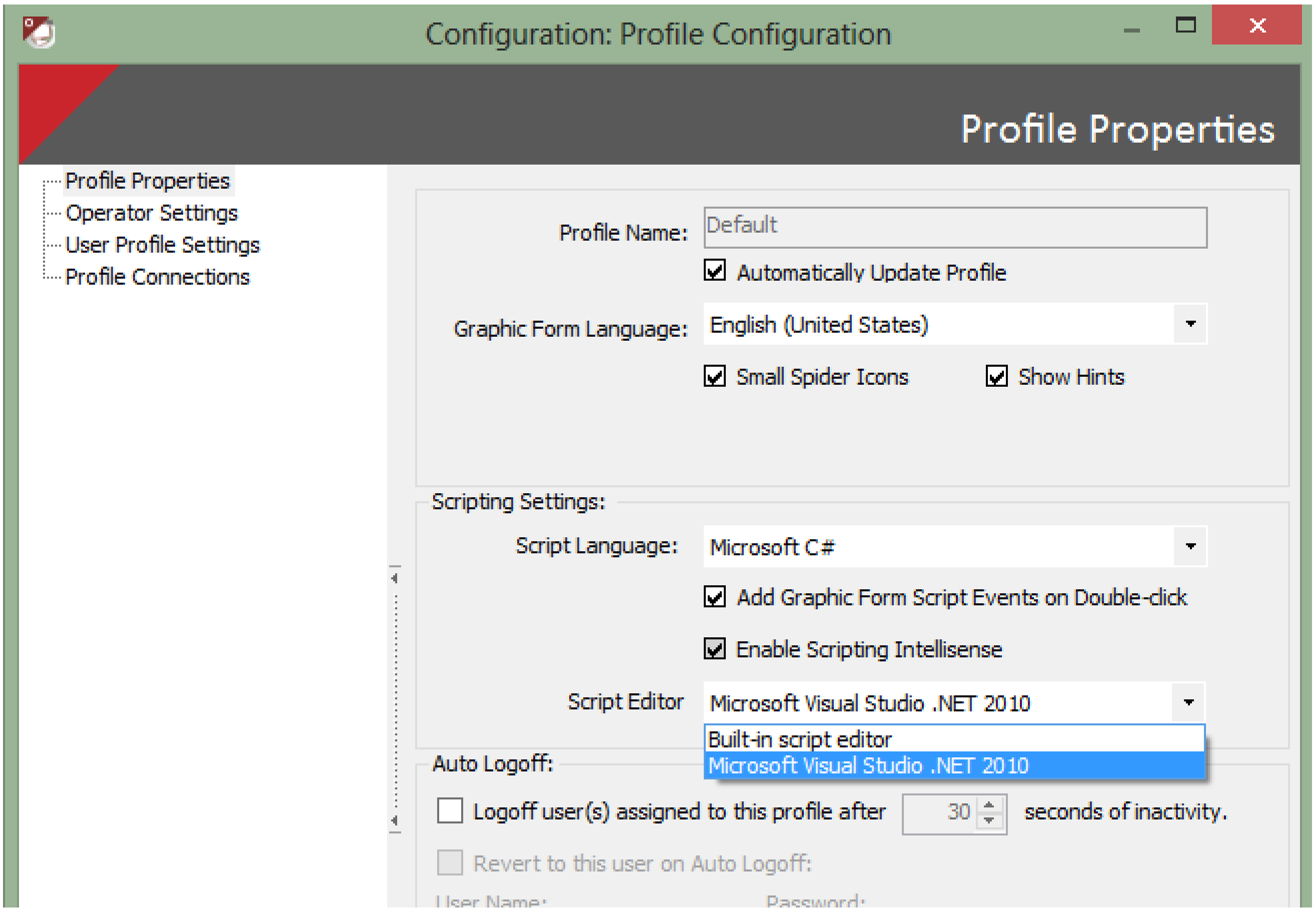
5. Getting started

This guide will use Microsoft Visual Studio as the editor. To configure the Designer to use Microsoft Visual Studio instead of the Internal Editor:

1) Download and install Microsoft Visual Studio or Microsoft Visual Studio Express edition
2) Change the editor in the **profile,** under the **Management** section within the Designer:



If Visual Studio has been detected, it will be available for selection:



Configuration: Profile Configuration

Profile Properties

- Profile Properties
- Operator Settings
- User Profile Settings
- Profile Connections

Profile Name: Default

☑ Automatically Update Profile

Graphic Form Language: English (United States)

☑ Small Spider Icons ☑ Show Hints

Scripting Settings:

Script Language: Microsoft C#

☑ Add Graphic Form Script Events on Double-click

☑ Enable Scripting Intellisense

Script Editor Microsoft Visual Studio .NET 2010

Built-in script editor
Microsoft Visual Studio .NET 2010

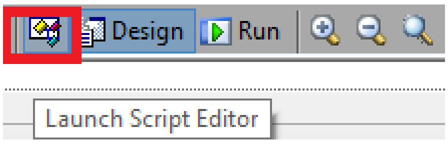
Auto Logoff:

□ Logoff user(s) assigned to this profile after 30 seconds of inactivity.

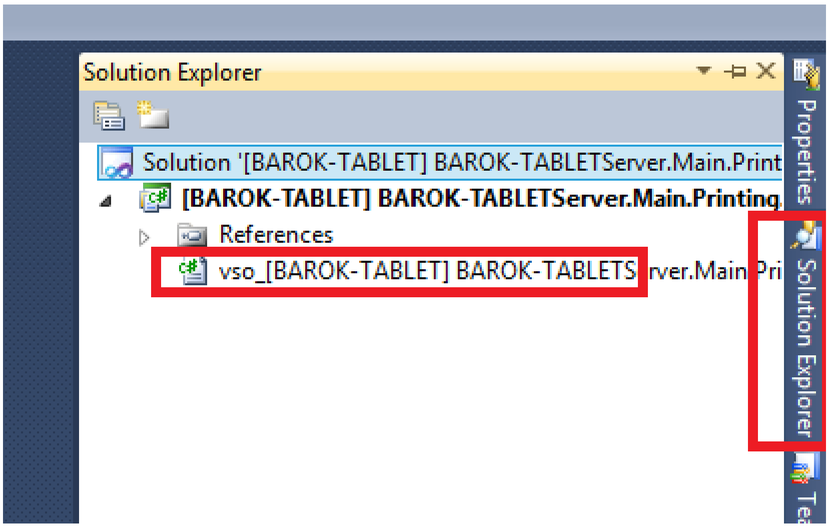
□ Revert to this user on Auto Logoff:

User Name: Password:

When clicking on the “Script” icon in the Designer, Visual Studio will now be used instead of the built-in editor:



Visual Studio will be loaded, with the “vso” or Visual Script Object now being available for edit from the Solution Explorer within Visual Studio:

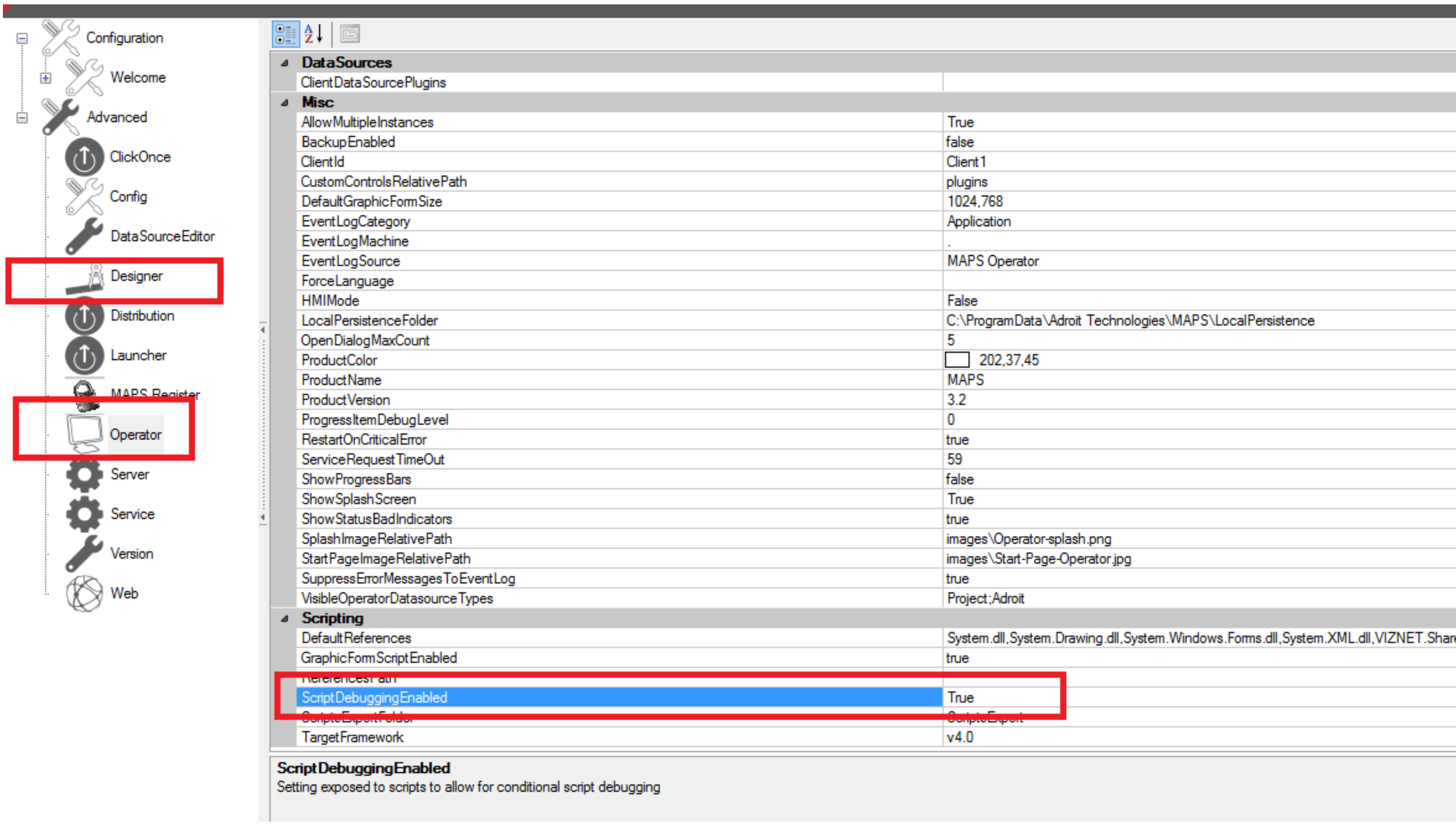

Solution Explorer
Properties
Solution '[BAROK-TABLET] BAROK-TABLETServer.Main.Print
[BAROK-TABLET] BAROK-TABLETServer.Main.Printing
References
vso_[BAROK-TABLET] BAROK-TABLETServer.Main.Pri
Solution Explorer
Tea

6. How to debug

A common requirement during scripting is to attach the Visual Studio debugging environment to the running process and debug certain functionality.

To enable debugging in the Designer or Operator:

1) Enable debugging output in the “Advanced” configuration section for either the Designer or Operator:

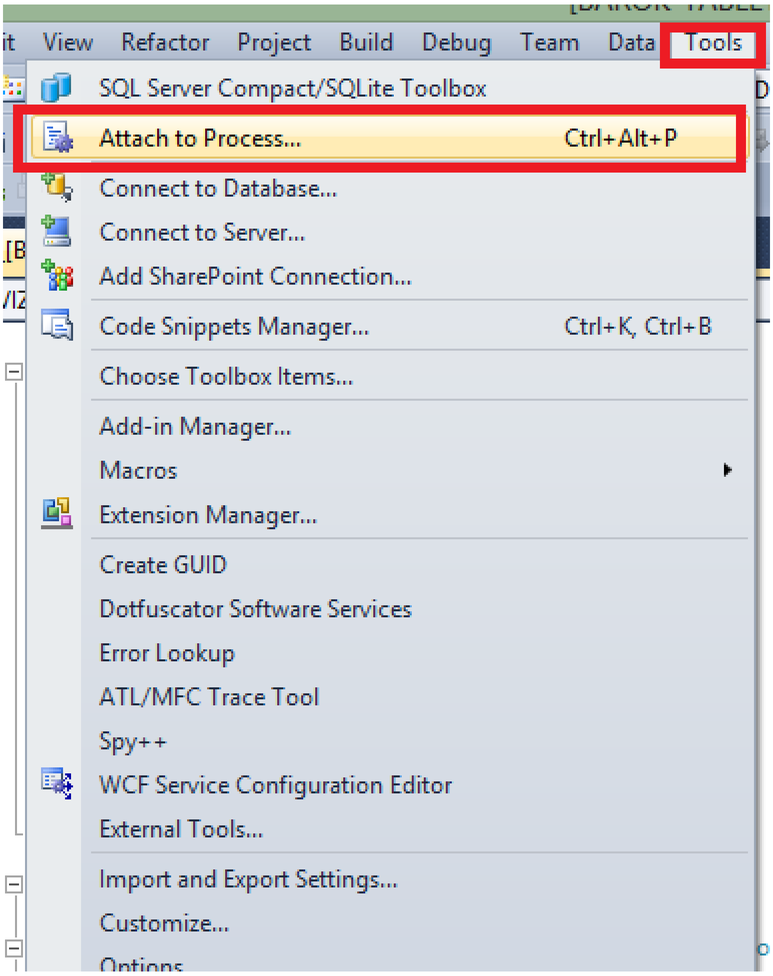


DataSources	
ClientDataSourcePlugins	
Misc	
AllowMultipleInstances	True
BackupEnabled	false
ClientId	Client1
CustomControlsRelativePath	plugins
DefaultGraphicFormSize	1024,768
EventLogCategory	Application
EventLogMachine	.
EventLogSource	MAPS Operator
ForceLanguage	
HMIMode	False
LocalPersistenceFolder	C:\ProgramData\Adroit Technologies\MAPS\LocalPersistence
OpenDialogMaxCount	5
ProductColor	□ 202,37,45
ProductName	MAPS
ProductVersion	3.2
ProgressItemDebugLevel	0
RestartOnCriticalError	true
ServiceRequestTimeOut	59
ShowProgressBars	false
ShowSplashScreen	True
ShowStatusBadIndicators	true
SplashImageRelativePath	images\Operator-splash.png
StartPageImageRelativePath	images\Start-Page-Operator.jpg
SuppressErrorMessagesToEventLog	true
VisibleOperatorDatasourceTypes	Project;Adroit
Scripting	
DefaultReferences	System.dll,System.Drawing.dll,System.Windows.Forms.dll,System.XML.dll,VIZNET.Shar
GraphicFormScriptEnabled	true
ReferencesPath	
ScriptDebuggingEnabled	True
ScriptsExportFolder	ScriptsExport
TargetFramework	v4.0

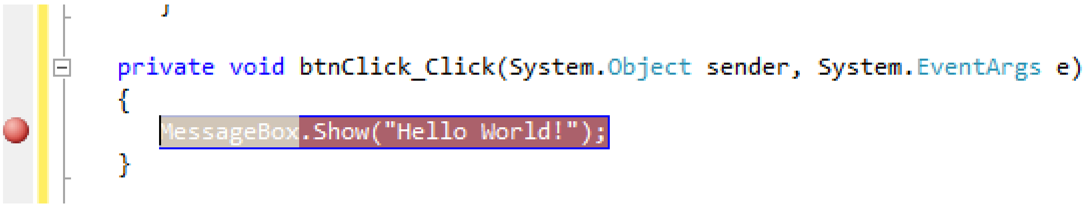
ScriptDebuggingEnabled
Setting exposed to scripts to allow for conditional script debugging

2) Restart the Designer/Operator to affect the configuration change.

3) Open the form to be debugged in the Operator/Designer. If using the Designer, ensure that the form is in "Run" mode.
4) Open Microsoft Visual Studio and attach to the process



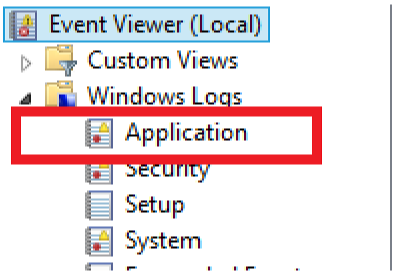
If the symbols are loaded, the breakpoints will resolve correctly and the code can be debugged:



```
    }

private void btnClick_Click(System.Object sender, System.EventArgs e)
{
    MessageBox.Show("Hello World!");
}
```

Other tips:

If the script fails in the Operator (due to an exception for example), this will be logged in the **Eventlog – Application** and can be viewed using the Windows Event Viewer**:**



7. Interaction

Interacting with Windows Controls follow the same basic paradigm as interaction for standard Windows Forms projects done in Visual Studio or Sharp Develop. The basic premise is setting properties of controls or attaching to their events and acting upon the event.

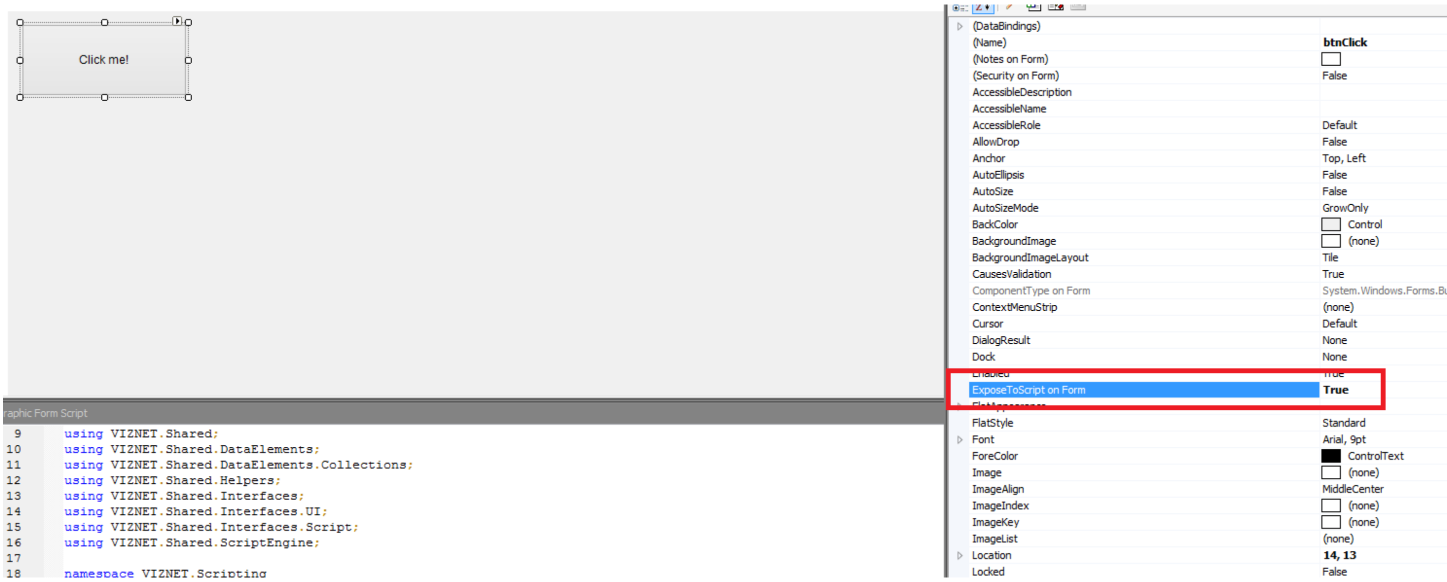
The following examples are also contained within a project that can be downloaded here: http://adroitdemo2.cloudapp.net/KBArticles/scripting_examples.zip

For example 7, the associated Microsoft Visual Studio 2010 sample solution can be downloaded here: http://adroitdemo2.cloudapp.net/KBArticles/CommonLibrary.zip

Example 1 – Setting the properties of a control

In this example we will change the ".Text" property of a Button (Windows Forms).

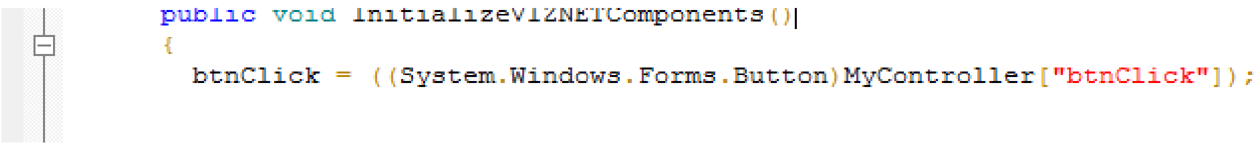
1) In the Designer, create a new form and place a button on the form
2) Expose the control to the scripting environment:



```
using VIZNET.Shared;
using VIZNET.Shared.DataElements;
using VIZNET.Shared.DataElements.Collections;
using VIZNET.Shared.Helpers;
using VIZNET.Shared.Interfaces;
using VIZNET.Shared.Interfaces.UI;
using VIZNET.Shared.Interfaces.Script;
using VIZNET.Shared.ScriptEngine;

namespace VIZNET.Scripting
```

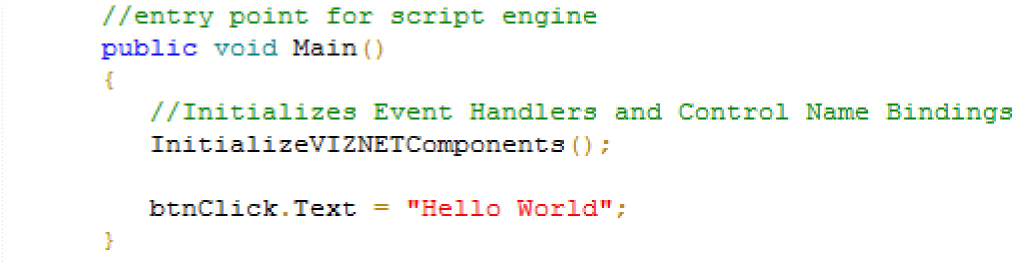
(DataBindings)	
(Name)	btnClick
(Notes on Form)	□
(Security on Form)	False
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoEllipsis	False
AutoSize	False
AutoSizeMode	GrowOnly
BackColor	□ Control
BackgroundImage	□ (none)
BackgroundImageLayout	Tile
CausesValidation	True
ComponentType on Form	System.Windows.Forms.Bu
ContextMenuStrip	(none)
Cursor	Default
DialogResult	None
Dock	None
Enabled	True
ExposeToScript on Form	True
FlatAppearance	
FlatStyle	Standard
Font	Arial, 9pt
ForeColor	■ ControlText
Image	□ (none)
ImageAlign	MiddleCenter
ImageIndex	□ (none)
ImageKey	□ (none)
ImageList	(none)
Location	14, 13
Locked	False

3) When you set this property to "True", a new line is added to the "InitializeComponents()" method



```
public void InitializeVIZNETComponents()
{
  btnClick = ((System.Windows.Forms.Button)MyController["btnClick"]);
```

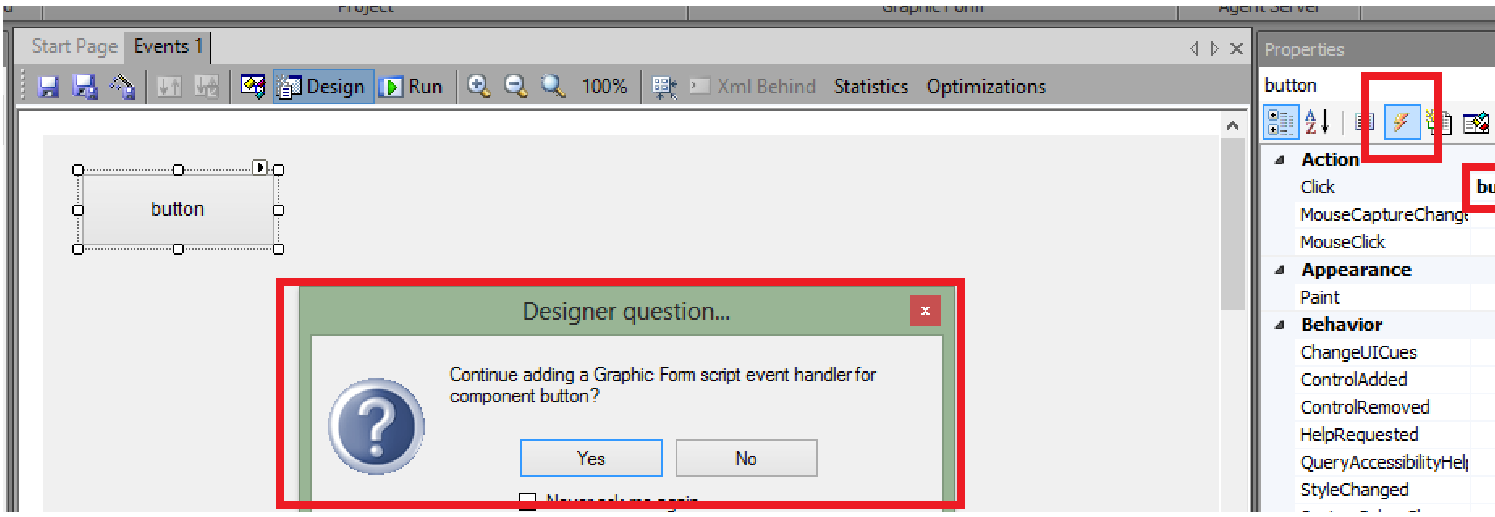
4) The “Main()” method of a script is always called when the form is initially loaded. In this example, we will simply change the “Text” property of the button when the form is loaded:



```
//entry point for script engine
public void Main()
{
   //Initializes Event Handlers and Control Name Bindings
   InitializeVIZNETComponents();

   btnClick.Text = "Hello World";
}
```

Example 2 – Attaching to the events of a control

In this example we will attach to the “Click” event of a button.



Designer question...

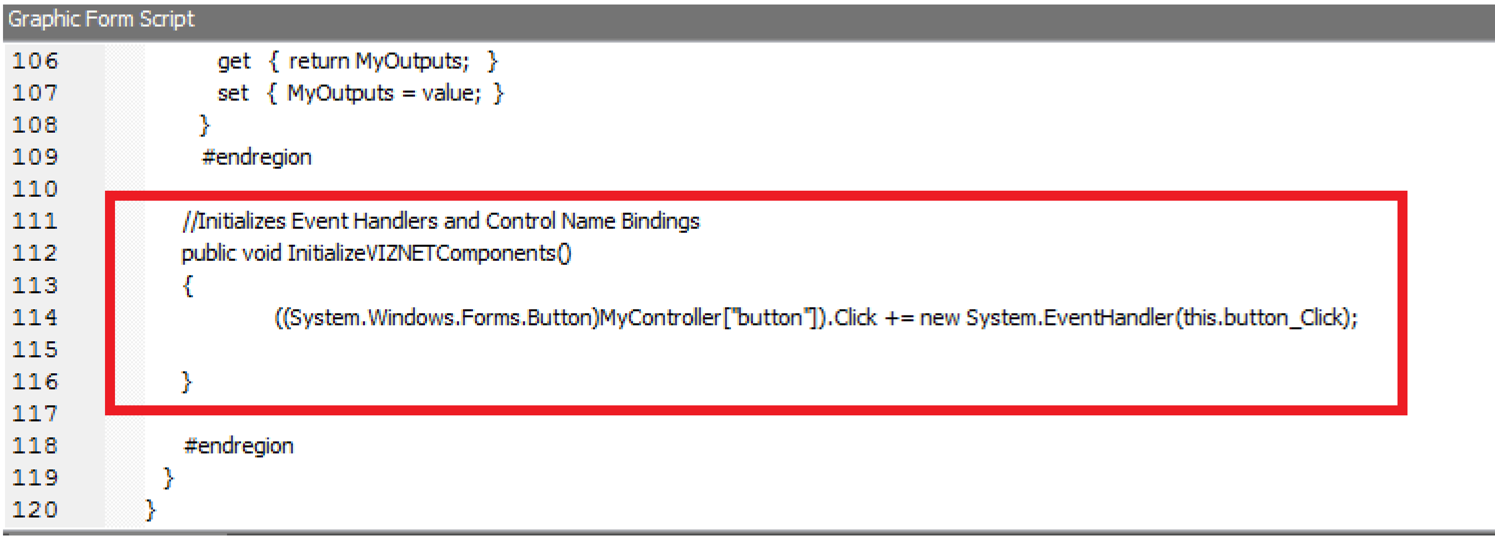
Continue adding a Graphic Form script event handler for component button?

Yes   No

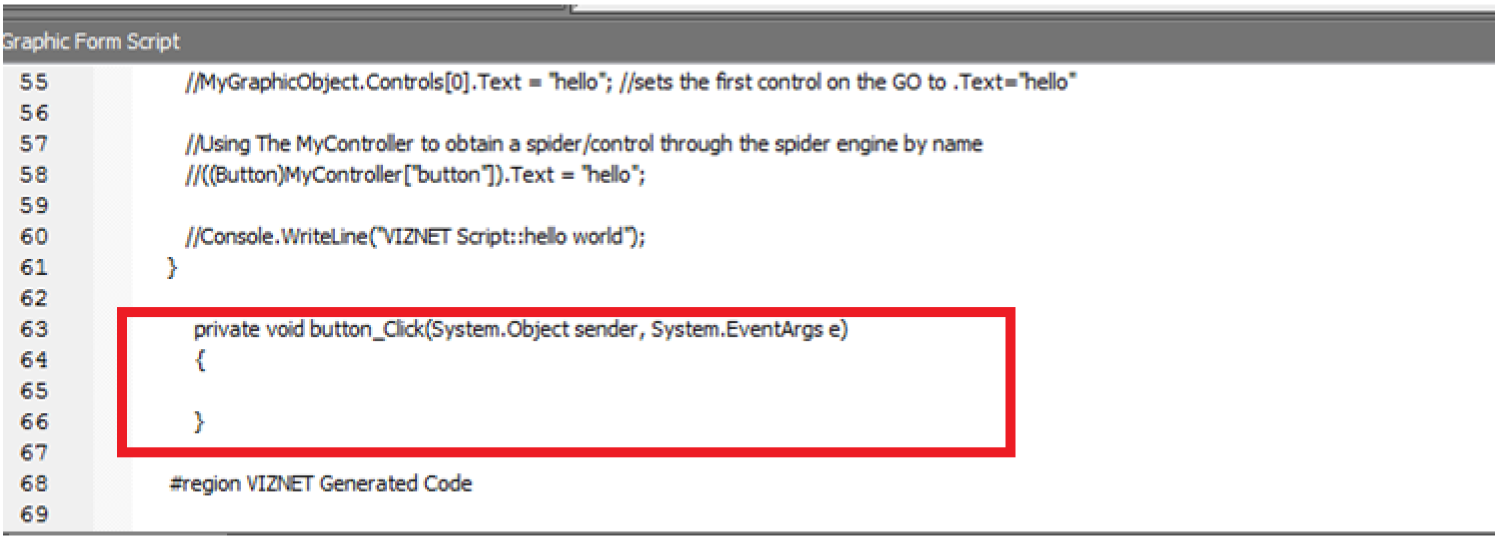
□ Never ask me again

The events of a control is listed in the “Properties” tool window under the Events section (highlighted above).

1) Each control provides various events that can execute a script when the event triggers.
2) Double Click on the event that you want to attach to.
3) A question will appear asking the user whether a script event handler should be added. Click “Yes”.



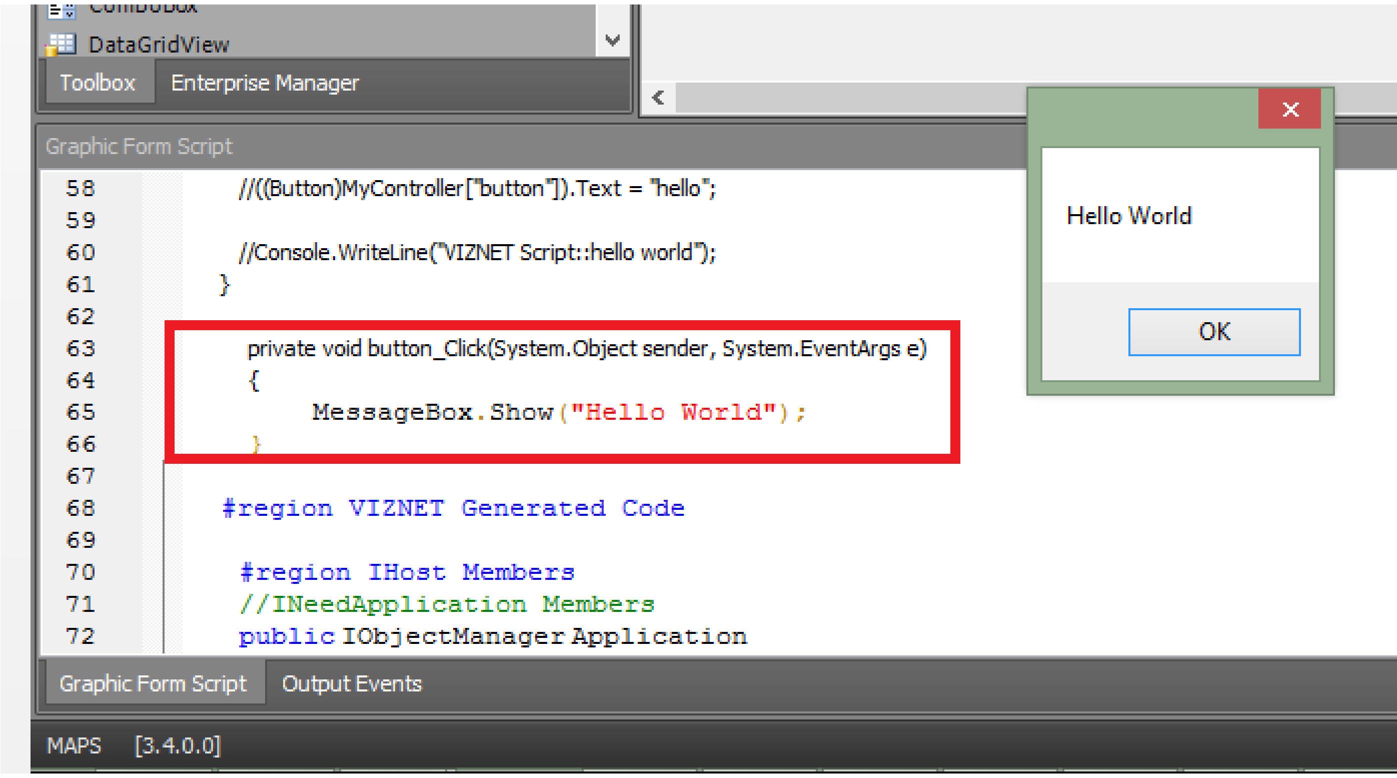
```
            get  { return MyOutputs;  }
            set  { MyOutputs = value; }
          }
          #endregion

        //Initializes Event Handlers and Control Name Bindings
        public void InitializeVIZNETComponents()
        {
                ((System.Windows.Forms.Button)MyController["button"]).Click += new System.EventHandler(this.button_Click);

        }

        #endregion
      }
    }
```



```
        //MyGraphicObject.Controls[0].Text = "hello"; //sets the first control on the GO to .Text="hello"

        //Using The MyController to obtain a spider/control through the spider engine by name
        //((Button)MyController["button"]).Text = "hello";

        //Console.WriteLine("VIZNET Script::hello world");
      }

        private void button_Click(System.Object sender, System.EventArgs e)
        {

        }

      #region VIZNET Generated Code
```

The script handler will automatically be inserted in the graphic form script.

Once the script handler have been inserted, go ahead and add the required functionality to the method. In this case we will add a simple Message Box:

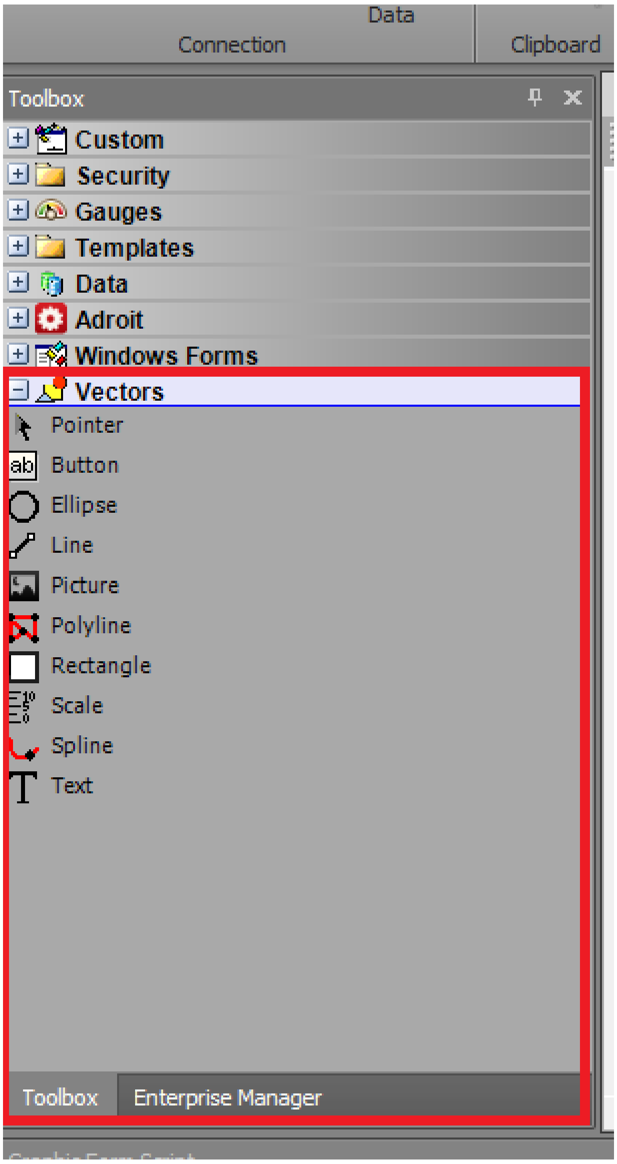


```
        //((Button)MyController["button"]).Text = "hello";

        //Console.WriteLine("VIZNET Script::hello world");
    }

        private void button_Click(System.Object sender, System.EventArgs e)
        {
            MessageBox.Show("Hello World");
        }

    #region VIZNET Generated Code

     #region IHost Members
     //INeedApplication Members
     public IObjectManager Application
```

Hello World

Example 3 – Changing the properties of a vector

Vectors are located in the toolbox under the “Vectors” section:



Vectors are “special” controls as they have the ability to zoom, rotate and scale, which other controls do not have. Vectors are often used to represent the animated part of a process because of these abilities.

To manipulate vectors within a script, you need a reference to **VIZNET.UI.Shared.DLL;** Vectors are also called “Canvas Objects” hence the reason why vector names start with “CO” when created.

In this example we will change the background color of the vector:

Add the following namespace to the top of the script:

```
using VIZNET.UI.Shared.GradientTypeEditor.Model;
```

For a **solid green** color:

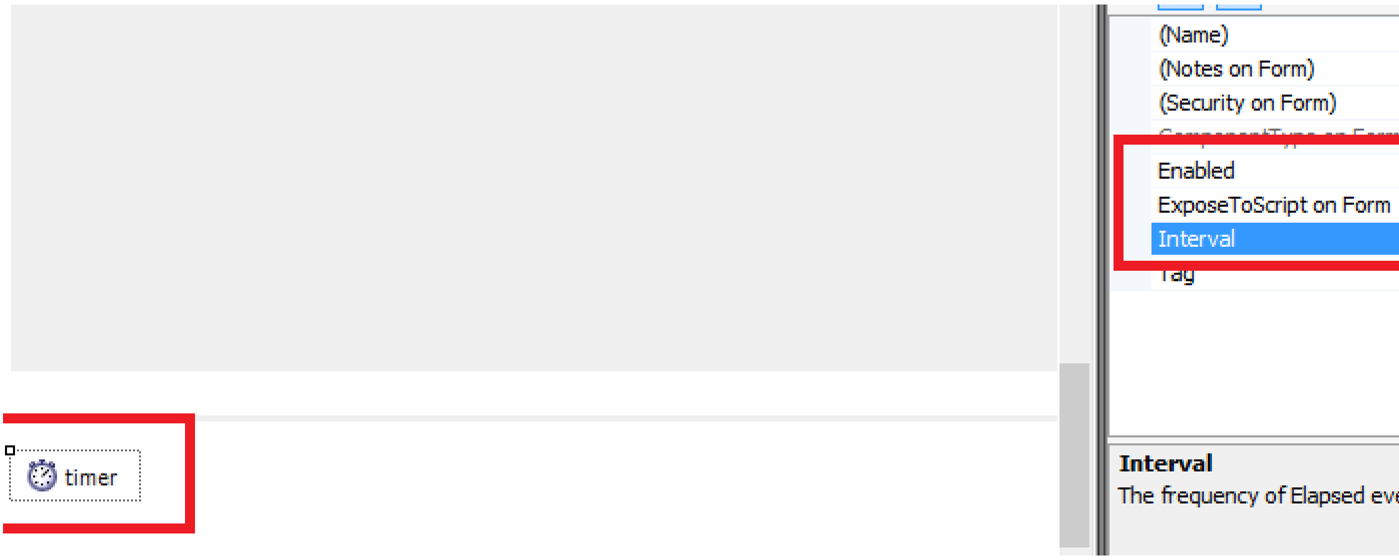
```
private void btnSolid_Click(System.Object sender, System.EventArgs e)
{
// solid color green
(cORectangle.BackColor as GradienColorSettings).ColorStyle = BrushType.Solid;
(cORectangle.BackColor as GradienColorSettings).SolidColor = Color.Green;
}
```

For a **green gradient** color:

```
private void btnGradient_Click(System.Object sender, System.EventArgs e)
{
// gradient with 3 colors i.e. white at position 0%, light green at position 50% and green at
position 100%
(cORectangle.BackColor as GradienColorSettings).ColorStyle = BrushType.Gradient;

ColorPoint[] __colorPoints = new ColorPoint[3];
__colorPoints[0] = new ColorPoint(Color.White, 0.0f);
__colorPoints[1] = new ColorPoint(Color.LightGreen, 0.5f);
__colorPoints[2] = new ColorPoint(Color.Green, 1.0f);

(cORectangle.BackColor as GradienColorSettings).ColorPositionsArray = __colorPoints;

}
```

Example 4 – Knowing when a form has finished loading

In various scenarios, it is important to know when a form has finished loading. For example, if you would like to attach to a specific event of a control after the form has completed the loading process, or change properties of the form itself. The safest way to ensure that the form has completed loading is to make use of a timer. The timer control will start running once the form has completed loading.



```
       /// <summary>
       /// This method will run as soon as the form has completed loading - triggered from a timer
tick
       /// </summary>
       /// <param name="sender"></param>
       /// <param name="e"></param>
        private void timer_Tick(System.Object sender, System.EventArgs e)
       {
             //Remember to disable the timer after the initial tick
             timer.Enabled = false;
             MessageBox.Show("Form load complete...");

       }
```

Example 5 – Changing the properties of the graphic form

In some cases certain properties of a form needs to change, for example the form title, or the form size, based on template information. To change the properties of a form at runtime, the **DesignerContainer** needs to be retrieved. The DesignerContainer *Parent* control is the actual control that hosts the form:

```
       // get the graphic form called "Form", if you change the name of the graphic form from "Form"
to something else then use that name instead
      DesignerContainer dc = MyController["Form"] as DesignerContainer;
      dc.ParentForm.Size = new Size(300, 300); // set the size of the graphic forms parent form
      dc.ParentForm.Text = "Hello World!"; // set the text of the graphic forms parent form
```

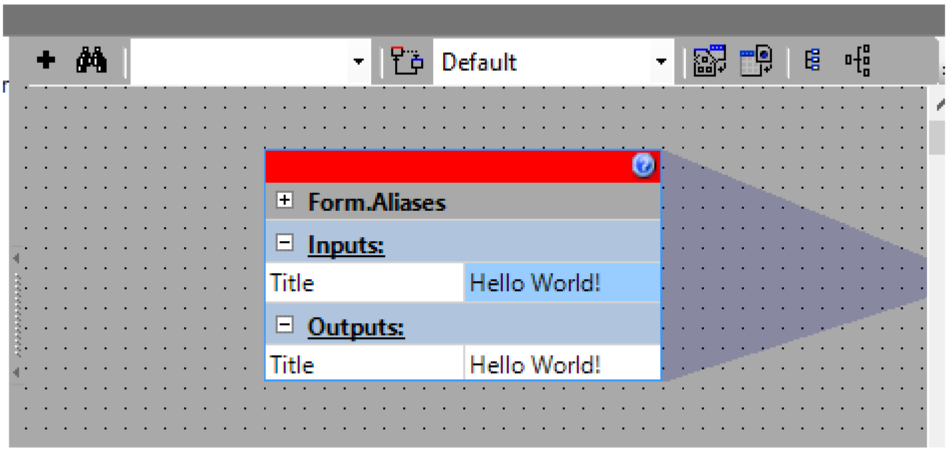
Now when the form has completed loading (using Example 4), the above code can change the properties of the form.

Example 6 – Changing the form title based on alias values

Aliases are a very common way of sending information between forms, for example to open a *Template*. The *template values* can then be used to update parameters and properties within the graphic form.

In this example, we will update the *Text* property of the form with the alias named: “Title”

Note: Remember to ensure that these aliases actually exist on the form.



Title	Hello World!

Title	Hello World!

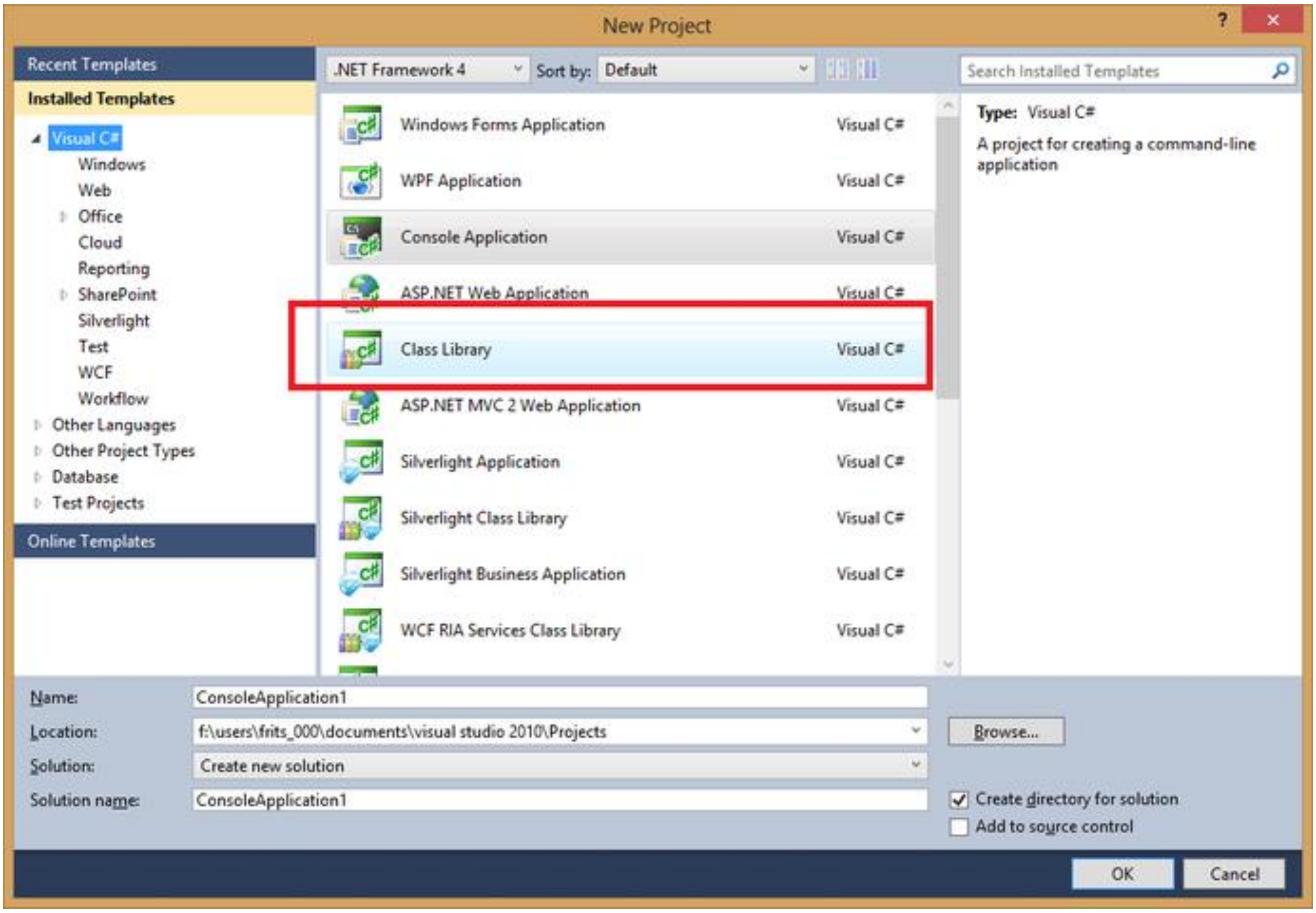
```
       //Get the alias we are interested in: (remember to reference 
VIZNET.Spiders.EventHashtableProperty.dll)
       VIZNET.Spiders.Event.EventHashtablePropertySpider aliasSpider = MyController["Form.Aliases"] 
       as VIZNET.Spiders.Event.EventHashtablePropertySpider;
       IOItem item = aliasSpider.OutputIOItems["Title"] as IOItem;
       if (item == null)
       {
          MessageBox.Show("Could not find alias");
          return;
       }
       DataElement de = item.Value;
       //Get the value of the actual data element
       string aliasValue = de.Value as String;

       // get the graphic form called "Form", if you change the name of the graphic form from "Form" 
to something else then use that name instead


        DesignerContainer dc = MyController["Form"] as DesignerContainer;
       dc.ParentForm.Text = aliasValue; // set the text of the graphic forms parent form
```

Example 7 – Sharing information between forms/Global Client Side variables

In many projects it is useful to share client side information between forms. This could include the previous graphic forms that has been opened by a user, specific project based information for that user etc.

To share information between forms it is possible to use a *Singleton* class that resides within a user created DLL. A *Singleton* is the name of a *Design Pattern* in which only one instance of the class is created (and can therefore easily be shared amongst graphic forms).

1) Create a *class library* project (this could be C#, VB.Net or any other .NET language you choose for your class library, the example provided here is in C#):


New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
ASP.NET Web Application Visual C#
Class Library Visual C#
ASP.NET MVC 2 Web Application Visual C#
Silverlight Application Visual C#
Silverlight Class Library Visual C#
Silverlight Business Application Visual C#
WCF RIA Services Class Library Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for creating a command-line application

Name: ConsoleApplication1

Location: f:\users\frits_000\documents\visual studio 2010\Projects

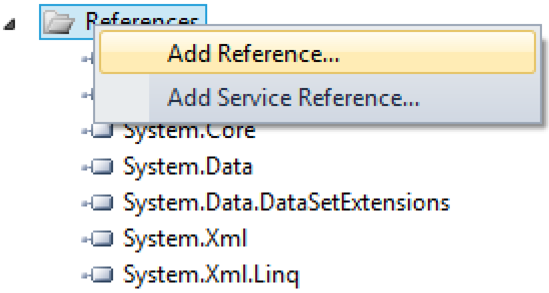
Solution: Create new solution

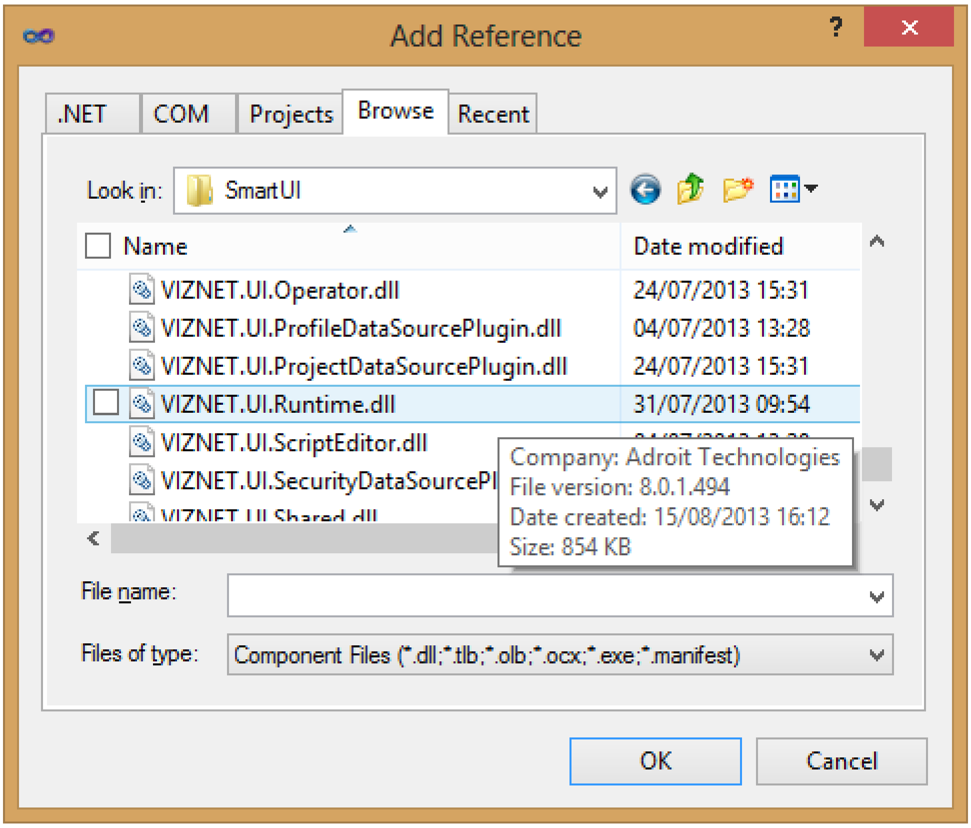
Solution name: ConsoleApplication1

☑ Create directory for solution

□ Add to source control

2) If you would like to use existing SmartUI references, remember to add them:





Look in: SmartUI

Name	Date modified
VIZNET.UI.Operator.dll	24/07/2013 15:31
VIZNET.UI.ProfileDataSourcePlugin.dll	04/07/2013 13:28
VIZNET.UI.ProjectDataSourcePlugin.dll	24/07/2013 15:31
□ VIZNET.UI.Runtime.dll	31/07/2013 09:54
VIZNET.UI.ScriptEditor.dll	
VIZNET.UI.SecurityDataSourcePl	
VIZNET.UI.Shared.dll	

Company: Adroit Technologies
File version: 8.0.1.494
Date created: 15/08/2013 16:12
Size: 854 KB

File name:

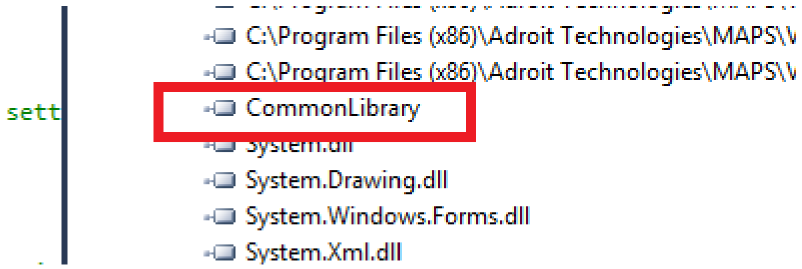
Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

In this example, we will share a simple string variable:

```
public class Shared
{
    //A reference to our only instance of this class
    private static Shared _shared;

    public String SharedString = string.Empty;

    public static Shared GetInstance()
    {
        if (_shared == null)
        {
            _shared = new Shared();
        }
        return _shared;
    }
}
```

Now we can use this library within any form script, by adding the reference and accessing the class:

First we add a reference to our common library inside the graphic form script:



sett

C:\Program Files (x86)\Adroit Technologies\MAPS\\
C:\Program Files (x86)\Adroit Technologies\MAPS\\
CommonLibrary
System.dll
System.Drawing.dll
System.Windows.Forms.dll
System.Xml.dll

And we have the ability to share information using the same instance of the class, across graphic forms or within the same graphic form:

```
 private void btnChange1_Click(System.Object sender, System.EventArgs e)
 {
CommonLibrary.Shared.GetInstance().SharedString = "Hello World";
 }

 private void btnChange2_Click(System.Object sender, System.EventArgs e)
 {
MessageBox.Show("The shared variable is: " + CommonLibrary.Shared.GetInstance().SharedString);

 }
```

Example 8 – Sharing re-usable functionality

Similar to the above example, various methods can also be added to the common library. For example, the ability to quickly read a data element or set a group of elements. The underlying mechanism of sharing the class instance and making use of the *Singleton* design pattern stays the same.

Example 9 – Read data elements from a data source

The “MyConnection” class provides various methods for interacting with data sources on the Server, these include reading from and writing to elements.
Data Elements travel between the server and client inside a DataElement Collection. The DataElement, DataElementCollection and DataElementEngine classes work together as part of the *Engine/Collection/Class* Design Pattern.

To read elements from a Data Source:

```
 try
{
   //Read the element "Adroit.systeminfo.second" from the Adroit Data Source
   DataElement de = DataElementEngine.NewDataElement("Adroit.systeminfo.second");
   DataElementCollection dec = DataElementEngine.NewDataElementCollection();
   dec.Add(de);
   MethodReturnInfo info = MyConnection.ReadDataElements(dec);
   if (!info.Success)


 {
      MessageBox.Show("Error reading element, the error was: " + info.ReturnInformation);
```

```
            return;
        }
        DataElementCollection returnDEC = info.ReturnObject as DataElementCollection;
        DataElement returnDE = returnDEC["Adroit.systeminfo.second"];
        MessageBox.Show("The value of: " + returnDE.Name + " is: " + returnDE.Value.ToString());
    }
    catch (Exception ex)
    {
        MessageBox.Show("An exception occurred: " + ex.ToString());
    }
```

Example 10 – Update the value of data elements in a data source

```
     try
    {
        //Update the element "Adroit.Analog.A1.value" in the Adroit Data Source
        DataElement de = DataElementEngine.NewDataElement("Adroit.Analog.A1.value");
        de.Value = 50.0;
        DataElementCollection dec = DataElementEngine.NewDataElementCollection();
        dec.Add(de);
        MethodReturnInfo info = MyConnection.UpdateDataElements(dec);
        if (!info.Success)
        {
            MessageBox.Show("Error updating element, the error was: " + info.ReturnInformation);
            return;
        }
        MessageBox.Show("Successful update");
    }
    catch (Exception ex)
    {
        MessageBox.Show("An exception occurred: " + ex.ToString());
    }
```

Example 11 – How to open a graphic form, from within a script

To open a form, and control various other client side functionality, the “ObjectManager” class can be used. In the example below, we read the graphic form (also a type of Data Element) on the Server, and then instruct the ObjectManager to open the form:

```
     try
    {
        //Fetch the graphic form, from the Server for a project called Samples, graphic form name:
Sample2
        string gfName = "Scripting examples.Example 11";
        DataElement de = DataElementEngine.NewDataElement(gfName);
         //this indicates that, if the form does not exist in the current folder, it will search in
     other folders of the project, if not found,   the server will search for an alternate match
     in other projects as well.
        de.Status = DataElement.DataElementStatus.SEARCHFORALTERNATES;
        DataElementCollection dec = DataElementEngine.NewDataElementCollection();
        dec.Add(de);


     MethodReturnInfo info = MyConnection.ReadDataElements(dec);
        if (info.Success)
        {
            DataElementCollection returnDec = info.ReturnObject as DataElementCollection;
            DataElement returnDe = returnDec[gfName] as DataElement;
            GraphicObject go = returnDe.Value as GraphicObject;
                   //Load this form with the default template set and level, alongside the alias
       value we have created - NOTE: WILL ONLY WORK IN THE OPERATOR
            ObjectManager.GetInstance().OpenGraphicObject(go, GraphicObjectDialogType.ModalForm,
"Level", "Default");
         }
         else
         {
            MessageBox.Show("An error occurred trying to retrieve graphic form");
         }
      }
      catch (Exception ex)
      {
         MessageBox.Show("An exception occurred: " + ex.ToString());
      }
```

Example 12 – How to open a graphic form with aliases, from within a script

```
try
      {
         //Fetch the graphic form, from the Server for a project called Samples, graphic form name:
Sample2
         string gfName = "Scripting examples.Example 12";
         DataElement de = DataElementEngine.NewDataElement(gfName);
         //this indicates that, if the form does not exist in the current folder, it will search in
other folders of the project, if not found, the server will search for an alternate match in other
projects as well.
         de.Status = DataElement.DataElementStatus.SEARCHFORALTERNATES;
         DataElementCollection dec = DataElementEngine.NewDataElementCollection();
         dec.Add(de);
         MethodReturnInfo info = MyConnection.ReadDataElements(dec);
         //We are now going to open a form and set one alias value. The alias name on the form we
are going to open is "Tag" and we will set the value to "Hello World";
         EventHashtable aliases = new EventHashtable();
         aliases.Add("Title", "Hello World"); // Set the alias name: "Tag", with the value: "Hello
World" for the form we are going to open
         if (info.Success)
         {
            DataElementCollection returnDec = info.ReturnObject as DataElementCollection;
            DataElement returnDe = returnDec[gfName] as DataElement;
            GraphicObject go = returnDe.Value as GraphicObject;
            //Load this form with the default template set and level, alongside the alias value we
have created - NOTE: WILL ONLY WORK IN THE OPERATOR
            MyApplication.OpenGraphicObject(go, GraphicObjectDialogType.ModalForm, "Level",
"Default", aliases);
         }
         else
         {
            MessageBox.Show("An error occurred trying to retrieve graphic form");
         }
      }
      catch (Exception ex)
      {
         MessageBox.Show("An exception occurred: " + ex.ToString());
      }
```