[Calling .NET from PowerBuilder](http://blogs.artinsoft.net/Mrojas/archive/2009/03/03/Calling-NET-from-PowerBuilder.aspx)

Most of our clients come from a medium-size to a big enterprise level. In these scenarios is very common to have different department using different technologies to solve their business needs.

These different technologies can be on a very homogeneous platform like .NET where you can easily interact between your VB.NET and C# assemblies, or they could be on differente technologies like ASP, Classic VB, C++, or Powerbuilder.

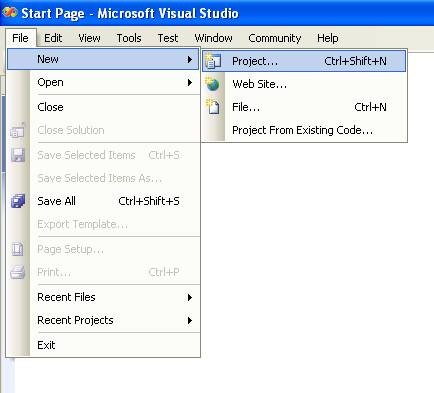
This post is about PowerBuilder, and in order to interact with PowerBuilder I think the easiest way is to expose your assemblies thru COM Interop.

So if there is some .NET functionality that you want to expose to PowerBuilder you just need to expose that functionality with a class in a ClassLibrary project with COM attributes.

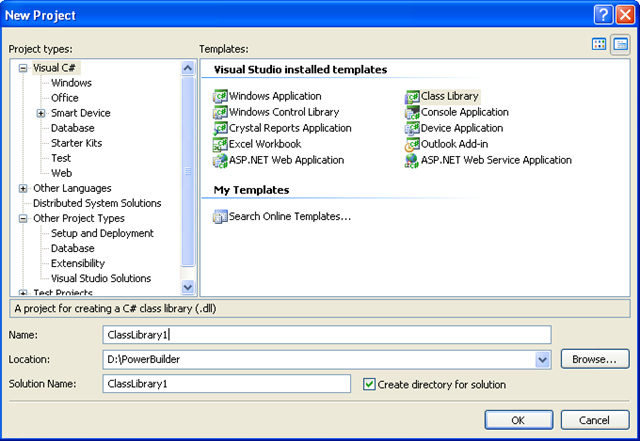
Let’s begin with a simple program to show how to comunicate Powerbuilder with C#.

NOTE:  If you don’t have Powerbuilder you can get a trial version from: <http://www.sybase.com/detail?id=1052162>

1. Open Microsoft Visual Studio
2. On the File Menu, choose the New option, and in the File submenu choose Project….

**Figure 1. Visual Studio File Menu. Choosing the option for a new project**

When you choose that option a dialog window will shown with the available options for new projects. In the option for C# Projects choose “Class Library”

**Figure 2. New Project dialog window**

You must introduce the new project name, location and solution name. Type something like ClassLibrary1, D:\PowerBuilder, ClassLibrary1.

1. When you finish creating your project you will have a code file called Class1.cs.
2. Change that code for something like :

using System;

using System.Collections.Generic;

using System.Text;

namespace SimpleClass {

public class Class1 {

public void SayHi() {

System.Windows.Forms.MessageBox.Show("Hello World!");

}

public String GiveMeDate() {

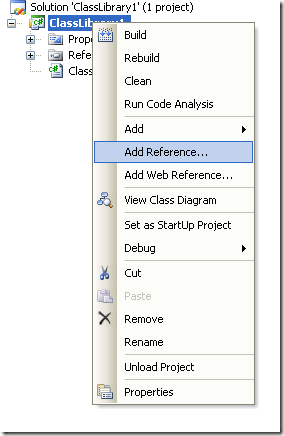
return DateTime.Now.ToLongDateString();

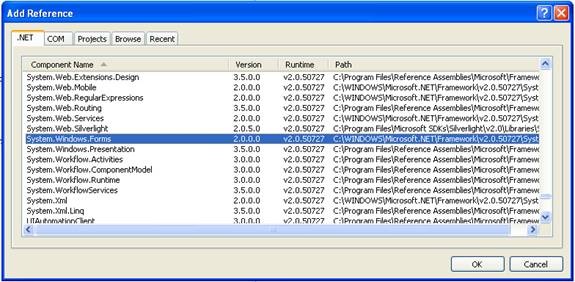
}

}

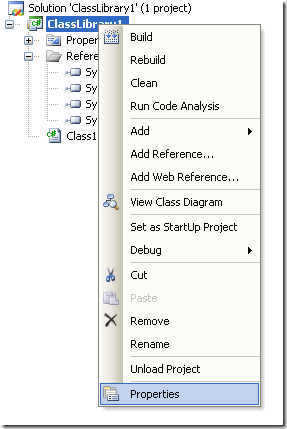
}

This will allow to test things like parameter passing, using different return types like strings or integers. But Before you continue you must add a reference to System.Windows.Forms to be able to use the MessageBox.

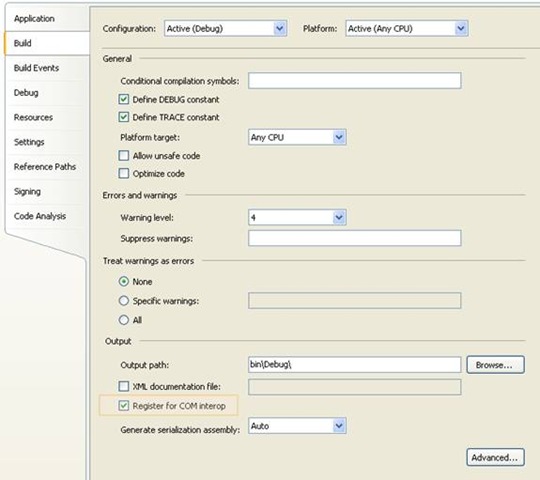
**Figure 3. Adding a reference**

**Figure 3. Reference to System.Windows.Forms**

1. Right click on the solution file and select properties:

**Figure 4. Option to change project properties**

1. Select the Register for COM Interop  checkbox

**Figure 5. Project properties window**

1. Return to Class1.cs code file
2. Add an using statement after the existing using lines on Class1.cs file:

using System.Runtime.InteropServices;

1. Add the following attributes to the class:

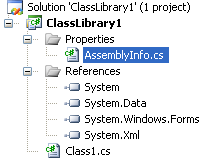
[ComVisible(true)]

[ClassInterface(ClassInterfaceType.AutoDual)]

[ProgId("ClassLibrary1.Class1")]

Note: the **ProgId** is very important, because these value will be use in PB to comunicate with the this code

1. Now you must edit the AssemblyInfo.cs

 **Figure 6. AssemblyInfo.cs File**

Now make sure to establish the COM settings in this file with statements like the following:

//Setting ComVisible to false makes the types in this assembly not visible

//to COM components. If you need to access a type in this assembly from

//COM, set the ComVisible attribute to true on that type.

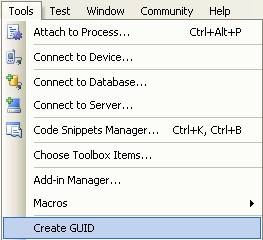
[assembly: ComVisible(true)]

// The following GUID is for the ID of the typelib if this project is exposed to COM

[assembly: Guid("69efac5b-d887-40f4-a7e9-2721ac3c1598")]

The Guid is also very important, because this is used to differentiate this component and it must be unique.

To generate a new GUID you can got to the Tools Menu and choose the option Create GUID

 **Figure 7. Option menu to create a GUID**

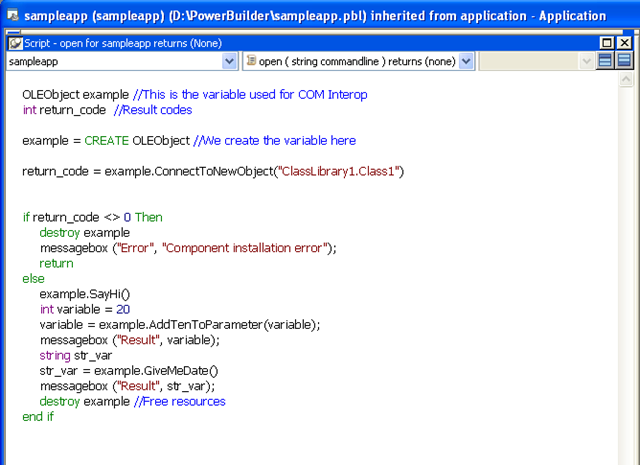
In the Create Guiddialog box, choose the fourth option and press Copy to put the contents on the Clipboard. Later, copy that value in the GUID attribute but remove the “{“ y “}”.

Now we are set. You only need to build the solution and the DLL. The build process with register the COM component.

If you will use the component on another computer you need to create an instalation program.

**Using your program from Powerbuilder**

Using your program from Powerbuilder is very easy. You just need code like the following:

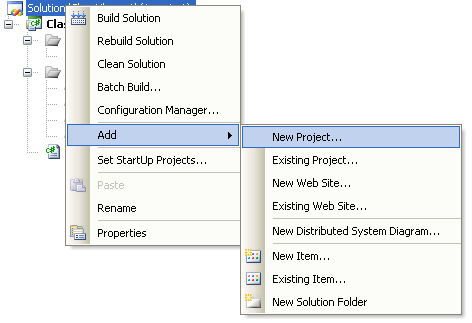
 **Figure 8. PB Code to call a C# class thru COM**

When you execute this program you will have 3 messageboxes :

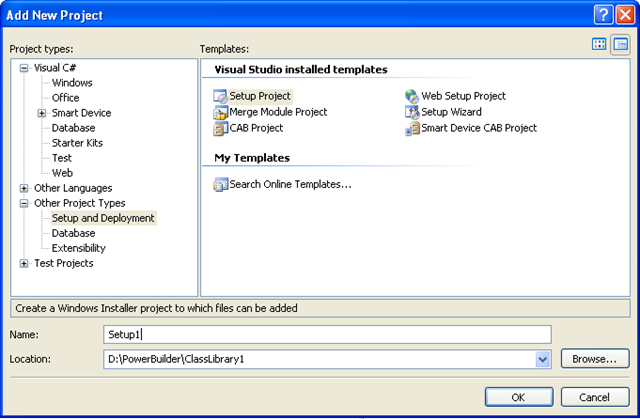
* Hello World!
* 30
* Monday, March 02, 2009 (this message will change depending of the day, locale and regional settings)

**Creating an instalation program**

1. Right click the solution and in the context menu choose Add and then new project.

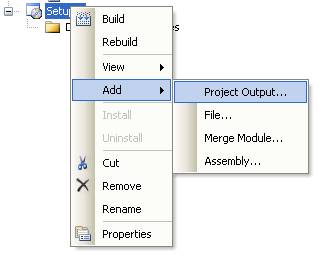
 **Figure 9. Context Menu to add a new project**

1. On the dialog box for Add New project, look for the Other Project Types section and the choose Setup Project.

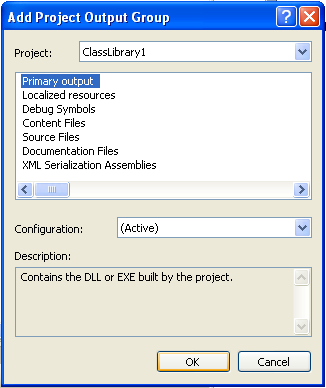
 **Figure 10. Creating a setup project**

In this dialog box indicate the name and location of the setup project. For example Setup1 and D:\Powerbuilder\ClassLibrary1.

1. Later, add a project to the setup program. To do that rigth click on the setup project and select Add, and in the submenu choose Project Output.

 **Figure 11. Adding a project to the setup project**.

1. A dialog box will be shown with a combo that allow you to select the proyects in the solution. Choose ClassLibrary1 and press OK.

 **Figure 12. Adding project output to the setup project.**

1. When you build this instalation program two files will be produced:

**Release**

D:\PowerBuilder\ClassLibrary1\Setup1\Release\Setup.exe

D:\PowerBuilder\ClassLibrary1\Setup1\Release\Setup1.msi

**Debug**

D:\PowerBuilder\ClassLibrary1\Setup1\Debug\Setup.exe

D:\PowerBuilder\ClassLibrary1\Setup1\Debug\Setup1.msi

When you run the instalation program, this program will handle the instalation of the .NET component and the COM registration.