

# SharePoint Development Setup

---

By RDA Software Engineer Herman Ip and RDA Project Manager Greg Galipeau

## 1. Introduction

Developers who are new to SharePoint development all seem to have encountered one same question: What should a typical SharePoint project look like? Although SharePoint is based on ASP.NET, it also introduces several capabilities that are not found in ASP.NET. For example, developers have to deal with implementing custom Site Definitions, List, Features, etc. Moreover, the final product of a SharePoint implementation is usually a SharePoint Solution Package (WSP). Creating a deployment solution manually can be a challenging task.

This paper contains an example of a typical setup of a SharePoint project. The main purpose is to give developers new to SharePoint a starting point to create SharePoint projects in Visual Studio.

## 2. VPC Preparation

In order to develop a SharePoint application, SharePoint must be installed on the development workstation. Since SharePoint is a server application, it must be installed on a Windows Server platform. Unfortunately, most development workstation is not running a Windows Server operating system. The easiest way to get around this obstacle is to develop SharePoint application on a Virtual PC (VPC). The developers will create a VPC that is running a Windows Server operating system. SharePoint and Visual Studio will then be installed on that VPC image.

This paper assumes the reader has already got a VPC image with SharePoint and Visual Studio.NET installed. To learn how to create a VPC for SharePoint, please reference the following article.

<http://www.pptspaces.com/sharepointreporterblog/Lists/Posts/Post.aspx?ID=28>

### 3. WSP Generation

#### 3.1. WSP Builder

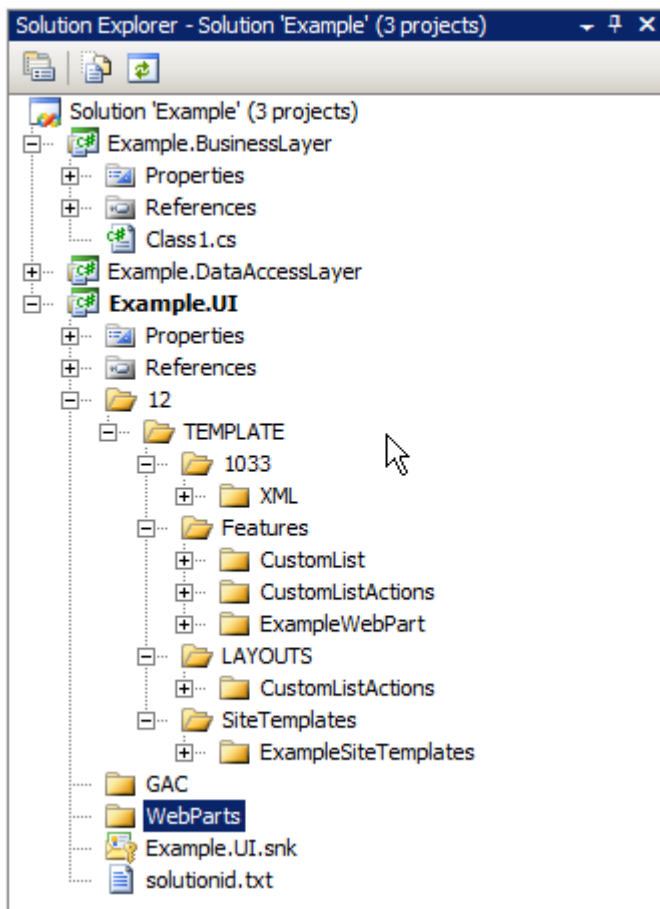
The best way to deploy a SharePoint application is to use a WSP. Creating a solution package from scratch can be a daunting task. There is a CodePlex project called WSP Builder that helps developers to create a WSP easily. WSPBuilder is a Visual Studio extension that allows developers to create a WSP through Visual Studio. WSPBuilder can be downloaded from the following link:

<http://www.codeplex.com/wspbuilder/Release/ProjectReleases.aspx?ReleaseId=15552>

The solution package generated by WSPBuilder takes care of adding <SafeControl> entry to the web application's web.config file and assemblies to the GAC. It just requires two things:

- All files to be installed to SharePoint's system folder (12\TEMPLATE) must be organized accordingly in its project hierarchy.
- All assemblies to be installed to GAC must be added to the GAC folder of the deployment project.

Below is a diagram that shows a sample of WSPBuilder based deployment project.



Files that are supposed to be added to the 12 hive are organized accordingly in the project. A GAC folder is also created to store assemblies to be installed in the GAC.

### 3.2. Visual Studio Extensions for Windows SharePoint Services

Another way to create a solution package is to use “Visual Studio Extensions for Windows SharePoint Services” (VSeWSS). It is a downloadable extension to Visual Studio from Microsoft that provides many valuable utilities to ease WSS development. For example, it provides templates to allow developers to create list definitions and event handlers easily. It also allows a WSP file to be created behind the scene.

Although both VSeWSS and WSP Builder can be used to create WSP file, WSP Builder is actually a better tool used to generate a solution package. Here are the reasons:

1. VSeWSS creates a WSP file behind the scene and gives developers only little control on the build process. With WSP Builder, the developer is creating the entire 12 hive for SharePoint. Although this requires the developers to have more knowledge about SharePoint, it provides the developers full control on what will be packaged in the WSP file.
2. WSP Builder provides other project templates for a solution such as workflow.
3. WSP Builder allows a WSP file to be created through command line other than the user interface.

Because of these reasons, this article will focus on using WSP Builder for deployment rather than VSeWSS.

## 4. Different types of SharePoint Projects

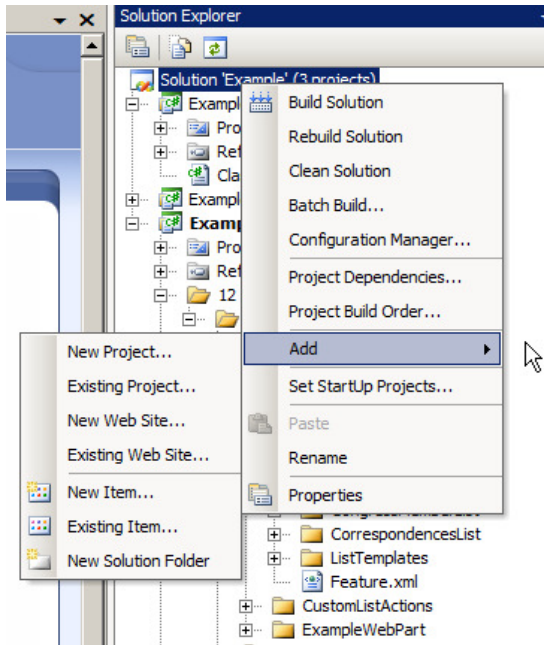
In a snapshot, a typical SharePoint application can be categorized in 3 different types of projects:

1. SharePoint specific – This type of project contains code that is used for SharePoint specific features. This includes Site Definitions, custom Content Type, custom List Definitions, Features, etc. This should be a WSPBuilder project.
2. Business Layer – This project looks just like a typical Business Layer project in ASP.NET.
3. Data Access Layer – This project looks just like a typical Data Access Layer project in ASP.NET.

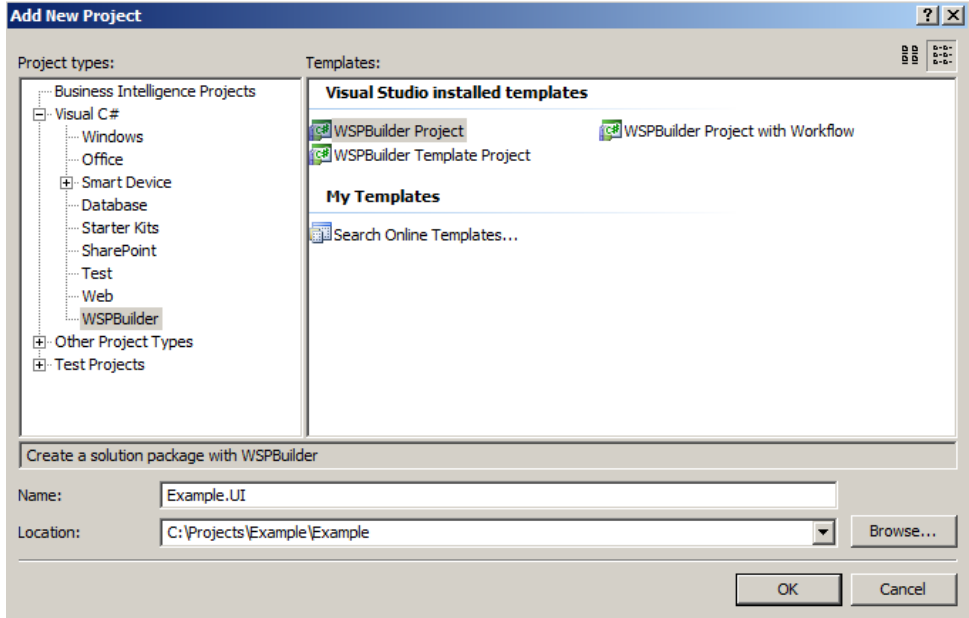
This is a typical 3-tier client/server application development setup where the SharePoint project represents the presentation layer. Three tier architecture increases flexibility, maintainability and reusability of an application.

### 4.1. SharePoint Project

As mentioned in earlier section, a SharePoint project should be using the WSPBuilder project template. If WSPBuilder is installed, a WSPBuilder project template will be available. The following snapshots show how to create a WSPBuilder project. In Solution Explorer, right click on the solution and select **Add->New Project...**



After that, the **Add New Project** dialog will be displayed. Under Visual C#, click on **WSPBuilder**, and double click on **WSPBuilder Project**.



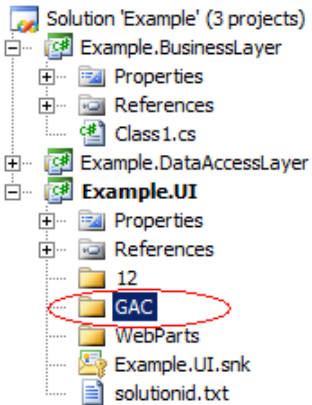
On the **Add New Project** dialog, enter the name for the project. Enter a name for this new project. In the example, the name of the project is called Example.UI.

**4.1.1. XML Files**

A SharePoint project is used to implement SharePoint specific features such as Site Definitions, List Definitions, Features, etc. These files are written in XML and should be defined under the 12\Template folder. WSPBuilder will automatically package all files under the 12 hive into the solution file.

### 4.1.2. Compiled Assembly

Some SharePoint files are implemented in Visual Basic.NET or C# and are compiled into an assembly. These include Feature Event Receivers, Site Provisioning Event Receivers, Web Parts, etc. All assemblies that are to be installed to the GAC must be copied to the GAC folder under the WSPBuilder project.



To automate this process, developer can take advantage of the project's Post-Built event by adding the following script to both the Debug and Release configuration:

```
xcopy $(TargetPath) $(ProjectDir)GAC /s /Y
```

### 4.2. Business Layer and Data Access Layer Projects

These projects use the Class Library project template and the products of these projects are assemblies. If these assemblies are to be installed in the GAC, they must be copied to the GAC folder under the WSPBuilder project. The following example shows the script to be specified in the Post-Build event for the Business Layer.

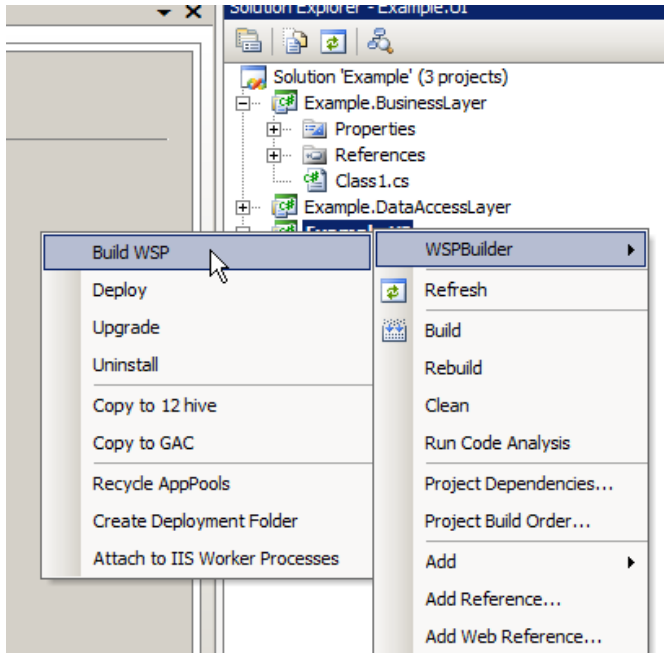
```
xcopy $(TargetPath) $(SolutionDir)Example.UI\GAC /s /Y
```

The above script copies the assembly to the GAC folder under the Example.UI project.

## Deployment

The section discusses how to create and deploy a deployment package using WSPBuilder. WSPBuilder provides an easy way to create a deployment package through the user interface in the Visual Studio.

In the Solution Explorer, right click the WSPBuilder project, and select **WSPBuilder->Build WSP**. WSPBuilder will then automatically create a wsp file.



Once a wsp file is created, the solution package can then be deployed on the local development machine by selecting **WSPBuilder -> Deploy**. If it is the first time the solution package is deployed, WSP Builder will add the solution and then deploy it. If the solution has already been deployed, WSPBuilder will upgrade the solution directly.

## 5. Conclusion

By now, developers new to SharePoint development should have a basic idea on what typical SharePoint projects look like. The example illustrates the projects layout of a typical 3-tier client/server that covers most of the projects usually found in SharePoint development.