Principio del formulario



|  |  |
| --- | --- |
| http://i.msdn.microsoft.com/Platform/Controls/PrinterFriendlyBanner/resources/msdn.gif | ©2009 Microsoft Corporation. All rights reserved. |

Development Tools and Techniques for Working with Code in Windows SharePoint Services 3.0 (Part 2 of 2)

**Summary:** Explore Windows SharePoint Services solutions, solution architecture, and techniques for creating, deploying, maintaining, and upgrading Windows SharePoint Services solutions. This article is part 2 of 2. (26 printed pages)

Patrick Tisseghem, [U2U](http://www.u2u.net/) [ http://www.u2u.net/default.aspx ]

June 2007

**Applies to:** Windows SharePoint Services 3.0, Visual Studio 2005 Extensions for Windows SharePoint Services 3.0

**Contents**

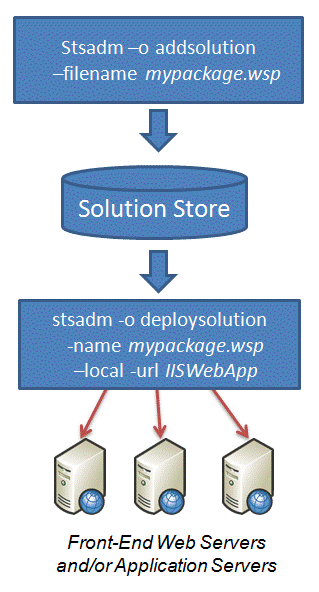
* [Deploying Windows SharePoint Services Solutions](http://msdn.microsoft.com/en-us/library/bb530301(printer).aspx#WSSDevToolsTechs_P2_DeployingWSSSolutions)
* [Manually Packaging a Windows SharePoint Services Solution](http://msdn.microsoft.com/en-us/library/bb530301(printer).aspx#WSSDevToolsTechs_P2_ManuallyPackWinSharePointServSolution)
* [Code Access Security and Web Part Solutions](http://msdn.microsoft.com/en-us/library/bb530301(printer).aspx#WSSDevToolsTechs_P2_CodeAccessSecurityandWebPartSolutions)
* [Managing Solutions](http://msdn.microsoft.com/en-us/library/bb530301(printer).aspx#WSSDevToolsTechs_P2_ManagingSolutions)
* [Conclusion](http://msdn.microsoft.com/en-us/library/bb530301(printer).aspx#WSSDevToolsTechs_P2_Conclusion)
* [About the Author](http://msdn.microsoft.com/en-us/library/bb530301(printer).aspx#WSSDevToolsTechs_P2_AboutAuthor)
* [Additional Resources](http://msdn.microsoft.com/en-us/library/bb530301(printer).aspx#WSSDevToolsTechs_P2_AdditionalResources)

This article is a continuation of [Development Tools and Techniques for Working with Code in Windows SharePoint Services 3.0 (Part 1 of 2)](http://msdn.microsoft.com/en-us/library/bb530302(printer).aspx) [ http://msdn.microsoft.com/en-us/library/bb530302.aspx ] .

# Deploying Windows SharePoint Services Solutions

The deployment of a Windows SharePoint Services solution to a front-end Web server or an application server is performed in two phases, as shown in Figure 19.

**Figure 19. Steps for deploying a SharePoint solution**



### Adding a Solution to the Solution Store

A SharePoint solution must be added to the solution store that is available in the server farm. The solution store is part of the configuration database and stores the various .wsp files. You add the solution to the store via the Stsadm.exe command-line utility or by using a programmatic approach.

**Using Stsadm.exe**

You can call Stsadm.exe with the **addsolution** operation by specifying the relative path to the .wsp file. Following is a sample.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl04');" \o "Copy Code)

stsadm –o addsolution –filename mysolution.wsp

|  |
| --- |
| **NoteNote:** |
| If you are localizing the solution, you can also use a third parameter: lcid. |

**Using the Windows SharePoint Services 3.0 Object Model**

In another approach, you can create a .NET Framework application that talks to the object model that is exposed by Windows SharePoint Services. You can add a SharePoint solution to the solution store by using one line of code as follows.

C#

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl06');" \o "Copy Code)

SPSolution solution = SPFarm.Local.Solutions.Add(@"C:\Package\MySol.wsp");

The **SPFarm** class in the **Microsoft.SharePoint.Administration** namespace that enables you to connect to the server farm (either one that is created locally or one you joined remotely). You can populate an **SPSolutionCollection** object with new solutions by calling the **Add** method, as long as it is either an instance of an **SPSolution** class or the path to the SharePoint solution file. An **SPSolution** instance is returned. Numerous properties and methods are exposed at this level.

### Deploying a Solution

Your next step is to actually deploy the solution on one or more Web servers that are Windows SharePoint Services–enabled, either as front-end Web servers or as application servers. You can deploy the solution in three ways:

* By using Stsadm.exe
* By using the Windows SharePoint Services 3.0 object model
* By using the SharePoint Central Administration Web Site

These deployment methods are described in the following sections.

**Using Stsadm.exe**

Stsadm.exe has an option, **deploysolution**, which you can use to deploy the solution via a command prompt. The name of the solution in the solution store is one of the parameters and the URL of the site collection you are targeting. The following is an example.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl07');" \o "Copy Code)

stsadm –o deploysolution –name mysolution.wsp

–url http://moss.litwareinc.com

Instead of targeting one site collection, you also have the option to push your solution to every site collection available within the server farm. You use the allcontenturls parameter for this as follows.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl08');" \o "Copy Code)

stsadm –o deploysolution –name mysolution.wsp –allcontenturls

By default, the solution is immediately deployed, but you can also schedule the deployment by using the time parameter.

The allowgacdeployment and allowcaspolicies parameters are also very important and are discussed later in more detail. In short, allowgacdeployment is **true** by default and enables Windows SharePoint Services to deploy the assemblies in the global assembly cache. The **allowcaspolicies** parameter enables the creation of a custom code access security (CAS) policy file and the activation of it in the Web.config file of the targeted site collection.

**Deploying a Solution via the Windows SharePoint Services 3.0 Object Model**

At the level of the **SPSolution** instance, you can use two methods to programmatically deploy your solution: **Deploy** and **DeployLocal**. Both can accept a collection of **SPWebApplication** objects populated with the Internet Information Services (IIS) Web applications that you want to target with the deployment. You populate the collection as follows.

C#

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl09');" \o "Copy Code)

Collection<SPWebApplication> webapps = new Collection<SPWebApplication>();

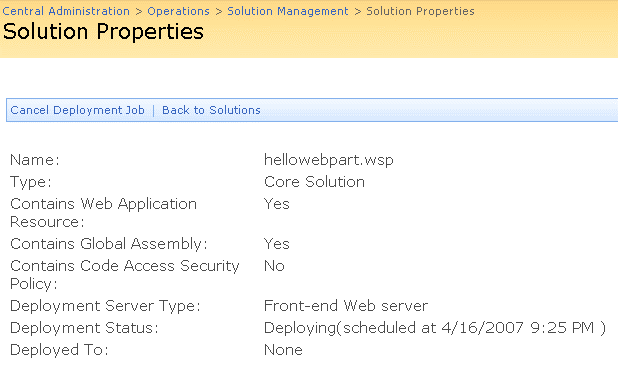
SPWebApplication webapp =

SPWebApplication.Lookup(new Uri("http://wss.litwareinc.com"));

webapps.Add(webapp);

Both methods also allow for the deployment of the assemblies included in the Web Part solutions to the global assembly cache. The difference between the two methods is that you can call **Deploy** with a DateTime parameter that stores the exact moment you want the solution to be deployed, as shown in Figure 20.

**Figure 20. SharePoint solution scheduled for deployment**



Following is a code example that uses the **Deploy** method.

C#

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl11');" \o "Copy Code)

solution.Deploy(DateTime.Now.AddMinutes(5), true, webapps, true);

Following is a sample of the call for the **DeployLocal** method.

C#

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl12');" \o "Copy Code)

solution.DeployLocal(true, webapps, true);

**Deploying a Solution via the SharePoint Central Administration Web Site**

Administrators can also navigate to the Operations page in Central Administration and use Solution Management to deploy the Windows SharePoint Services solution to an IIS Web application either immediately or at a specific time in the future.

Administrators receive a warning when the manifest file that is part of the solution requires assemblies to be deployed in the global assembly cache.

# Manually Packaging a Windows SharePoint Services Solution

Developers of Windows SharePoint Services solutions will very often create SharePoint solution packages in a manual way. This is the case when there is a need to:

* Deploy .NET assemblies in the private application folder instead of the global assembly cache.
* Add code access security permissions to the solution that must be applied during the deployment.
* Deviate from the names used by default for the Feature folders.
* Localize the Windows SharePoint Services solution.
* Associate Feature event handlers to certain types of Windows SharePoint Services solutions, such as Web Part solutions.
* Add resources (XML files, pictures, and so on) to the solution package.

To manually create a solution file, you perform these basic steps:

1. Collect all individual solution files in a folder. There are no concrete guidelines about how you should do this, but a best practice is to separate the different types of solution files into their own subfolders.
2. Create a .ddf file (abbreviation of Diamond Directive File) that defines the structure of the Windows SharePoint Services solution file. This file contains the list of individual solution files that determine the output .wsp file.
3. Execute the command-line utility MakeCab.exe with the .ddf file as input and the .wsp file as output.

The following short walkthrough demonstrates all of the steps.

### Walkthrough: Generating and Deploying a Custom Web Part Solution Package

Windows SharePoint Services 3.0 offers developers the option to execute custom code when a Feature is installed, activated, deactivated, or uninstalled. An example is a Web Part that is dependent on a specific task list. When the Web Part Feature is activated, custom code can check whether this task list is part of the lists within the site. If not, the code creates the list, and then removes the list when the Feature is deactivated. The custom code is wrapped into a .NET assembly referred to as the Feature Receiver Assembly.

This walkthrough assumes you have a Web Part project created, as discussed in [Development Tools and Techniques for Working with Code in Windows SharePoint Services 3.0 (Part 1 of 2)](http://msdn.microsoft.com/en-us/library/bb530302(printer).aspx) [ http://msdn.microsoft.com/en-us/library/bb530302.aspx ] . When the Web Part Feature is installed, activated, deactivated, or uninstalled, Windows SharePoint Services fires asynchronous events. You can handle these events in a custom .NET assembly by creating a .NET class that inherits from the abstract **Microsoft.SharePoint.SPFeatureReceiver** class. Following are the class from the sample and the four members to implement.

C#

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl14');" \o "Copy Code)

using System;

using System.Diagnostics;

using System.Collections.Generic;

using System.Text;

using Microsoft.SharePoint;

namespace MSDN.Samples

{

public class MSDNTaskListEventHandler: SPFeatureReceiver

{

public override void FeatureActivated(SPFeatureReceiverProperties properties)

{

SPSite sitecollection = (SPSite)properties.Feature.Parent;

SPWeb web = sitecollection.RootWeb;

try

{

// -- Check if list exists.

SPList list = web.Lists["MSDN Tasks"];

}

catch

{

// -- If not, create the list.

web.Lists.Add("MSDN Tasks", "A custom list", SPListTemplateType.Tasks);

}

}

public override void FeatureDeactivating(SPFeatureReceiverProperties properties)

{

SPSite sitecollection = (SPSite)properties.Feature.Parent;

SPWeb web = sitecollection.RootWeb;

try

{

// -- Check if list is there, and if so, delete it.

SPList list = web.Lists["MSDN Tasks"];

web.Lists.Delete(list.ID);

}

catch (Exception ex)

{

}

}

public override void FeatureInstalled(SPFeatureReceiverProperties properties)

{

}

public override void FeatureUninstalling(SPFeatureReceiverProperties properties)

{

}

}

}

The coding work results in two assemblies. One assembly contains the code delivering the Web Part. A second assembly contains the previous code. At the moment of writing, Visual Studio Extensions for Windows SharePoint Services 3.0 does not allow you to connect the event handler with the Web Part Feature definition file. In addition, Web Part assembly must be deployed in the private application folder instead of the global assembly cache. Because of this, you must create the solution package manually.

|  |
| --- |
| **NoteNote:** |
| In the following steps, the way you organize the different files representing the solution components can be adapted to your own preferences and can be part of the Visual Studio 2005 solution. |

Create a folder with two subfolders to collect all of the solution components. A first subfolder stores the assemblies (named Assemblies in this article), and the second subfolder stores the different XML files defining the Features (named Features in this article). Copy the Web Part assembly and the event handler assembly into the Assemblies folder.

Create a subfolder under the Features folder for every Feature that must be included in the SharePoint solution. There is only one Feature for this walkthrough. Assume that it is called **MSDNTaskCreator**; the Features folder contains a subfolder with that name. At the root of this folder, add a feature.xml file that contains the following XML.

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl16');" \o "Copy Code)

<Feature Title="MSDNTaskCreator"

Id="55312295-a323-4333-b875-1bbe8ef7fd04"

Description="Small Web Part creating a custom task item"

Version="1.0.0.0" Scope="Site" Hidden="FALSE"

ReceiverAssembly="MSDNFeatureEventhandlers, Version=1.0.0.0, Culture=neutral, PublicKeyToken=5e5a470a5445a8f1"

ReceiverClass="MSDN.Samples.MSDNTaskListEventHandler"

DefaultResourceFile="core" xmlns="http://schemas.microsoft.com/sharepoint/">

<ElementManifests>

<ElementManifest Location="elementManifest.xml" />

<ElementFile Location="MSDNTaskCreator.webpart" />

</ElementManifests>

</Feature>

How does this XML deviate from the XML that is generated with Visual Studio Extensions for Windows SharePoint Services 3.0? Two additional attributes are added to the feature.xml file:

* The **ReceiverAssembly** attribute contains the full strong name of the .NET assembly that contains the event handler code.
* The **ReceiverClass** attribute stores the full name of the class within that assembly.

You must create a manifest file in the root folder. It is different from the one that is generated by Visual Studio Extensions for Windows SharePoint Services 3.0. Following are the contents.

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl17');" \o "Copy Code)

<Solution SolutionId="d63d0395-96a4-449e-83ce-5f7239bbd3ad" xmlns="http://schemas.microsoft.com/sharepoint/" >

<FeatureManifests>

<FeatureManifest Location="MSDNTaskCreator\feature.xml" />

</FeatureManifests>

<Assemblies>

<Assembly Location="MSDNTaskCreator.dll" DeploymentTarget="WebApplication" >

<SafeControls>

<SafeControl Assembly="MSDNTaskCreator, Version=1.0.0.0, Culture=neutral, PublicKeyToken=9f4da00116c38ec5" Namespace="MSDN.Samples" TypeName="MSDNTaskCreator" Safe="True" />

</SafeControls>

</Assembly>

<Assembly Location="MSDNFeatureEventHandlers.dll" DeploymentTarget="GlobalAssemblyCache" />

</Assemblies>

</Solution>

Notice that the name of the Feature no longer includes a GUID. The first assembly element has an attribute named **DeploymentTarget**, with the value **WebApplication** instead of **GlobalAssemblyCache**. A second assembly element with the definition of the .NET assembly contains the event handler code to deploy in the global assembly cache.

Now we can create the .ddf file named, in this case, **.wsp\_structure.ddf**. Create it directly in the DeploymentFiles folder. First, add the following header information.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl18');" \o "Copy Code)

;

; \*\*\* .ddf file for generating SharePoint solution.

;

.OPTION EXPLICIT ; Generate errors

.Set CabinetNameTemplate=MSDNTaskCreatorWebPart.wsp

.set DiskDirectoryTemplate=CDROM ; All cabinets go in a single directory

.Set CompressionType=MSZIP;\*\* All files are compressed in cabinet files

.Set UniqueFiles="ON"

.Set Cabinet=on

.Set DiskDirectory1=Package

The header contains two fairly dynamic parts:

* **CabinetNameTemplate** is set to the name of the SharePoint solution file (MSDNTaskCreatorWebPart.wsp).
* **DiskDirectory1** is set to **Package**. This is the directory containing the generated .wsp file.

The second part of the .ddf file defines the structure of the package.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl19');" \o "Copy Code)

; \*\*\* the manifest file

manifest.xml manifest.xml

; \*\*\* the feature files

Features\MSDNTaskCreator\feature.xml MSDNTaskCreator\feature.xml

Features\MSDNTaskCreator\elementManifest.xml MSDNTaskCreator\elementManifest.xml

Features\MSDNTaskCreator\MSDNTaskCreator.webpart MSDNTaskCreator\MSDNTaskCreator.webpart

; \*\*\* the assemblies

Assemblies\MSDNTaskCreator.dll MSDNTaskCreator.dll

Assemblies\MSDNFeatureEventhandlers.dll MSDNFeatureEventhandlers.dll

The .ddf file is input for MakeCab.exe, a tool you can get by installing the Microsoft Cabinet SDK (C:\Program Files\Microsoft Cabinet SDK) and it is also part of the Smart Devices SDK (C:\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools).

|  |
| --- |
| **NoteNote:** |
| You can download the Microsoft Cabinet SDK from [Internet Client SDK: Microsoft Cabinet SDK](http://download.microsoft.com/download/platformsdk/cab/2.0/w98nt42kmexp/en-us/cabsdk.exe) [ http://download.microsoft.com/download/platformsdk/cab/2.0/w98nt42kmexp/en-us/cabsdk.exe ] . |

To facilitate the packaging and deployment, create a batch file with the following content.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl22');" \o "Copy Code)

set MakeCabTool=c:\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools\makecab.exe

set SPAdminTool=%CommonProgramFiles%\Microsoft Shared\web server extensions\12\BIN\stsadm.exe

"%MakeCabTool%" /f wsp\_structure.ddf

"%SPAdminTool%" -o addsolution -filename package\ MSDNTaskCreatorWebPart.wsp

"%SPAdminTool%" -o deploysolution -name MSDNTaskCreatorWebPart.wsp -immediate -allowGACDeployment -url http://moss.litwareinc.com

The first two lines are the settings of the paths to the Makecab and Stsadm command-line tools. Next, there is the line creating the solution package.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl23');" \o "Copy Code)

makecab.exe /f wsp\_structure.ddf

As a result of the execution, MSDNTaskCreatorWebPart.wsp appears in the Package folder. The next line adds the MSDNTaskCreatorWebPart.wsp to the solution store in your server farm by executing the following command.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl24');" \o "Copy Code)

stsadm.exe -o addsolution -filename Package\MSDNTaskCreatorWebPart.wsp

The final line in the batch file is the deployment of the solution to one of your site collections. You can use the Solution Management page under the **Operations** tab in Central Administration, or open a Command Prompt window again and execute the following command line.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl25');" \o "Copy Code)

stsadm.exe -o deploysolution -name MSDNTaskCreatorWebPart.wsp -local -allowGACDeployment -url http://moss.litwareinc.com

The Web Part Feature is installed but not activated. To activate the Feature, open the Site Collection Features page, and then click the **Activate** button. Because there is code that executes when the **FeatureActivated** event fires, the MSDN Task list is created. Deactivating this feature removes this task list from the root site of the site collection.

# Code Access Security and Web Part Solutions

In many locked-down environments, administrators do not allow custom code components to have Full trust. Administrators might choose to deploy a solution to the Web application \bin folder, where permissions must be specifically granted. Let’s look at the steps that are involved.

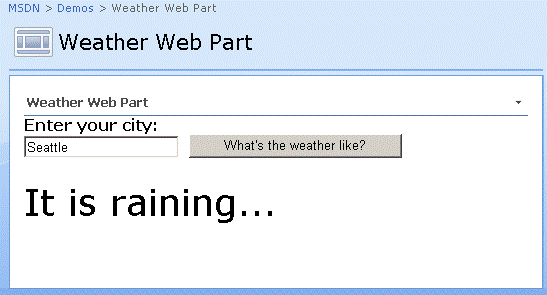
We can show all of this with a small Web Part that connects to a Web service that returns weather information for a specific city. If you build and deploy this Web Part by using Visual Studio 2005 Extensions for Windows SharePoint Services 3.0, the .NET assembly is deployed in the global assembly cache. You cannot intervene in this process on your development computer to configure the solution generation process and deployment differently. Because of the deployment in the global assembly cache, your Web Part gets full trust and does not have any security problems connecting to the Web service.

This scenario works if administrators allow your Web Part assemblies to be deployed in the global assembly cache. However, your Web Part assembly can often end up in the private \bin folder of the IIS Web application. As a Web Part developer, you then depend on the trust level set in the web.config file by administrators. In the end, you can encounter security problems with Web Parts that perform functions similar to our weather Web Part, as described in the following walkthrough.

### Walkthrough: Code Access Security and Web Part Solutions

Let’s assume you have a small Web service that returns the weather for a city, and that there is a Web Part that consumes the information, as shown in Figure 21. (You can work out an example with any type of Web service.)

**Figure 21. Web Part consuming a weather Web service**



To deploy the Web Part assembly in the private application folder of the IIS Web application instead of the global assembly cache (which is the default when using Visual Studio 2005 Extensions for Windows SharePoint Services 3.0), you can make a change in the manifest file that will enforce this. You can set the **DeploymentTarget** attribute at the level of the **Assembly** element to **WebApplication** instead of **GlobalAssemblyCache**, as shown in the following example.

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl27');" \o "Copy Code)

<Solution SolutionId="1de3b0fc-78e9-4fe6-ae63-51ea50109982" xmlns="http://schemas.microsoft.com/sharepoint/" >

<FeatureManifests>

<FeatureManifest Location="WeatherWebPart\feature.xml" />

</FeatureManifests>

<Assemblies>

<Assembly Location="WeatherWebPart.dll"   
 DeploymentTarget="WebApplication" >

<SafeControls>

<SafeControl Assembly="WeatherWebPart, Version=1.0.0.0, Culture=neutral, PublicKeyToken=9f4da00116c38ec5" Namespace="WeatherWebPart" TypeName="WeatherWebPart" Safe="True" />

</SafeControls>

</Assembly>

</Assemblies>

</Solution>

Next, you must create the Windows SharePoint Services solution manually. The following .ddf file shows how to package the different components that make up the Web Part solution.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl28');" \o "Copy Code)

.OPTION EXPLICIT

.Set CabinetNameTemplate=WeatherWebPart.wsp

.set DiskDirectoryTemplate=CDROM ; All cabinets go in a single directory

.Set CompressionType=MSZIP ;\*\* All files are compressed in cabinet files

.Set UniqueFiles="OFF"

.Set Cabinet=on

.Set DiskDirectory1=Package

manifest.xml manifest.xml

assemblies\WeatherWebPart.dll WeatherWebPart.dll

Features\WeatherWebPart\feature.xml WeatherWebPart\feature.xml

Features\WeatherWebPart\elementManifest.xml WeatherWebPart\elementManifest.xml

Features\WeatherWebPart\WeatherWebPart.webpart WeatherWebPart\WeatherWebPart.webpart

A simple call to Makecab.exe with the .ddf file as input generates the Windows SharePoint Services solution.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl29');" \o "Copy Code)

makecab.exe /f WeatherWebPart.ddf

You can add the solution to the solution store by executing the following command in a Command Prompt window.

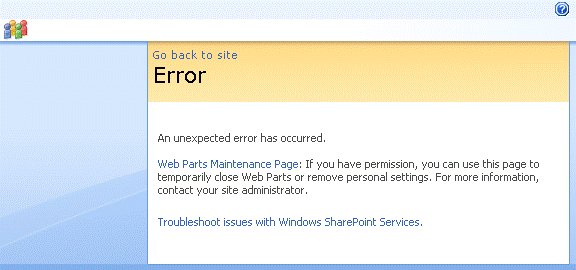
[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl30');" \o "Copy Code)

stsadm.exe -o addsolution -filename package\weatherwebpart.wsp

Now, navigate to Central Administration and the Solution Management page. From here you can deploy the solution. Notice that there is no warning because the manifest file does not require the assembly to be deployed in the global assembly cache. Proceed and deploy the solution to one of your IIS Web applications. It’s a good idea to verify in the physical folder that is associated with the IIS Web application (which is located, by default, in Inetpub\wwwroot\wss\VirtualDirectories\IIS Web application name) that the assembly is available in the \bin folder.

Assuming the trust level in the web.config file is not set to Full, you get an exception if you try to run your weather Web Part, as shown in Figure 22.

**Figure 22. Weather Web Part deployed in private application folder**



The Weather Web Part deployed in the private application folder causes unexpected problems. However, the error is not unexpected. Open the Windows Event Viewer (located in Administration Tools) and you find the full details of the error: "Request for the permission of type 'System.Net.WebPermission, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089' failed."

In other words, your Web Part is not granted the permission to communicate with the Web service. How do you solve this? One way is to raise the trust level in the web.config file to Full, but this is risky. From the moment you raise the trust level, all of the privately deployed assemblies get the same basic permissions as the assemblies that are deployed in the global assembly cache. A better solution is to request the permissions that are needed to run the Web Part correctly in SharePoint pages and include that demand inside the manifest file. Administrators who deploy the solution will receive a notification that specific requests for permissions are pending, and they can decide whether to grant these permissions. When proceeding, a copy is made of the policy file that is currently activated, and the requested permissions for the Web Part are added. This new policy file becomes the one activated in the web.config file. Now we can examine all of these steps in more detail.

One piece of information is already available. You have the full details of the required permission (discussed previously). Another piece of information is the full public key blob of the assembly you’re working with. To retrieve this information, open a Command Prompt window, and execute the following command.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl32');" \o "Copy Code)

Secutil.exe –hex –s WeatherWebPart.dll > keyblob.txt

The result is a text file with the full public key of the assembly in question. The tool used is secutil.exe, which is part of the .NET Framework SDK.

Next, open the manifest file, and add the following **CodeAccessSecurity** element (a good location is between the **FeatureManifests** and the **Assemblies** elements).

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl33');" \o "Copy Code)

<CodeAccessSecurity>

<PolicyItem>

<PermissionSet class="NamedPermissionSet" version="1"

Description="My webpart's permission set">

<IPermission class="AspNetHostingPermission" version="1"

Level="Minimal"/>

<IPermission class="SecurityPermission" version="1"

Flags="Execution" />

<IPermission version="1" Unrestricted="True"

class="Microsoft.SharePoint.Security.SharePointPermission, Microsoft.SharePoint.Security, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" />

**<IPermission class="System.Net.WebPermission, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" Unrestricted="True"**

**version="1">**

**<ConnectAccess>**

**<URI uri="$OriginHost$"/>**

**<URI uri="http://moss:95/webservices/.\*"/>**

**</ConnectAccess>**

**</IPermission>**

</PermissionSet>

<Assemblies>

**<Assembly Name="WeatherwebPart" Version="1.0.0.0" PublicKeyBlob="0x00240000048000009400000006020000002400005253413100040000010001000DAF8ED8D945CD2ABB2EE7953A6039B791A725F11B4588AC6D70B3E0648F955E9ED4C3C43CB044B8B0E8A6FF4D4FFBE9E3B9297D45F688A7264534E12414E17539305207EC961DA94DF294E7722CCD9BDBFC95A896E996F57156705D281EC39280BD604E87724556AF5807D146963F19F5B43DB69E1F22695463153A553260D2" />**

</Assemblies>

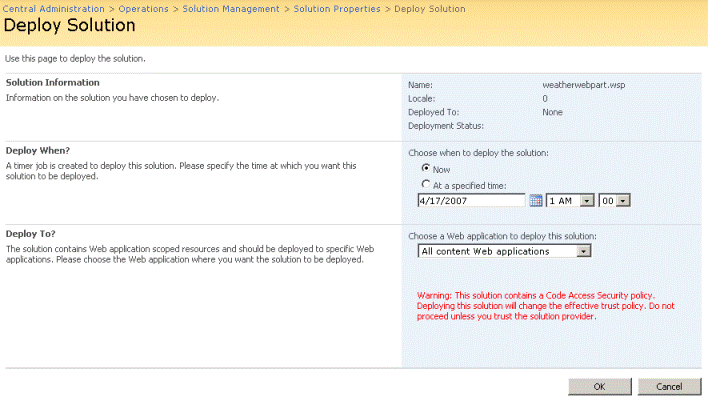
</PolicyItem>

</CodeAccessSecurity>

In the previous code, the **IPermission** and **Assembly** element areas are important to review. First, the **IPermission** element asks for permission to communicate with the Web service (we assume that this Web service is hosted on http://moss:95 IIS Web Application). Next, the **Assembly** element contains the details of the assembly in question: the name, version, and blob you must retrieve from the keyblob.txt file that is generated via the secutil.exe utility.

When these changes are applied to the manifest file, you must regenerate the Windows SharePoint Services solution and re-add it to the solution store. When you deploy the solution, you will notice a warning at the bottom of the page (see Figure 23) that indicates that the solution contains a code access security policy that will take effect if you continue to deploy the solution. If the administrators do not see a problem with this, they can continue and the Web Part is made available.

**Figure 23. Deploying a Web Part solution with a code access security policy**



If you followed all of the earlier steps correctly, the weather Web Part works again as before. Behind the scenes, you notice a new entry in the **securityPolicy** element of the web.config file that looks like the following.

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl35');" \o "Copy Code)

<securityPolicy>

<trustLevel name="WSS\_Medium" policyFile="C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\config\wss\_mediumtrust.config" />

<trustLevel name="WSS\_Minimal" policyFile="C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\config\wss\_minimaltrust.config" />

<trustLevel name="WSS\_Custom" policyFile="C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\config\wss\_custom\_wss\_minimaltrust.config" />

</securityPolicy>

The new level—WSS\_Custom—is now the active trust level in the web.config file.

# Managing Solutions

Windows SharePoint Services 3.0 also offers support for managing SharePoint solutions. You can retract solutions, remove solutions from the solution store, and upgrade solutions in several ways.

### Retracting Solutions

When you retract a SharePoint solution, Windows SharePoint Services physically removes all of the solution components from the location where they were deployed. Retracting a solution can be completed in three ways:

* By using Stsadm.exe
* By using Central Administration
* By using the Windows SharePoint Services 3.0 object model

These methods are described in the following sections.

**Using Stsadm.exe**

The option named **retractsolution** accepts several parameters. The name of the solution is a required parameter. Optionally, you can specify the URL for a specific IIS Web application and site collection, or remove the solution from all places where it was deployed via the allcontenturls parameter. The time parameter schedules the retraction for a job definition. You use the immediate parameter when you want to perform actions directly.

Following is a typical command for retracting a solution.

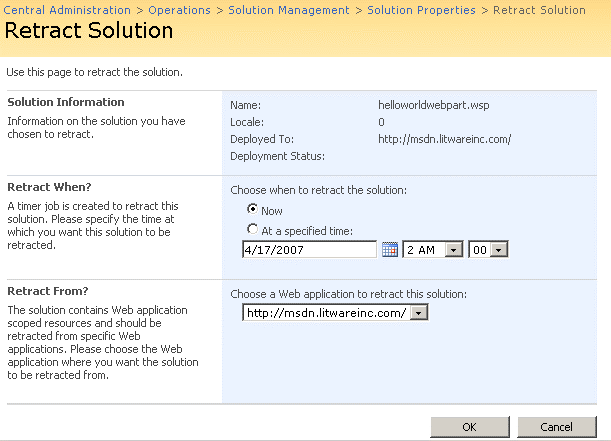
[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl36');" \o "Copy Code)

stsadm.exe –o retractsolution –name hellowebpart.wsp -immediate

**Using Central Administration**

After the solution is available in the solution store, you access it via the solution management page in Central Administration. From here you can start the process of retracting a solution, as shown in Figure 24.

**Figure 24. Retracting a Web Part from an IIS Web application**



**Using the Windows SharePoint Services 3.0 Object Model**

A final approach to retracting a solution is through the Windows SharePoint Services 3.0 object model. The **SPSolution** class exposes the **Retract** and **RetractLocal** methods. You can use **Retract** to schedule the retracting of the solution. Both methods offer the option to retract the solution from all or from only a specific collection of **SPWebApplication** objects. The following code example retracts a Web Part solution from all IIS Web applications that are available on a local computer.

C#

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl38');" \o "Copy Code)

SPSolution solution = SPFarm.Local.Solutions["hellowebpart.wsp"];

solution.RetractLocal();

**Retracting Web Part Solutions**

You can retract a solution that delivers Web Parts by using one of the previously mentioned options. However, you should notice that retracting the Web Part does not remove the .webpart entry from the Web Part gallery. As a consequence, the Web Part remains advertised in the **Add a Web Part** dialog box, and users can still see it. However, when users add the Web Part to a page, errors are shown because the Web Part is no longer registered as a safe control, and the assembly is removed from either the local bin folder or the global assembly cache. Also notice that retracting a Web Part solution causes existing Web Part instances to stop working and displays an error in the SharePoint pages. To solve this, you can have a deactivation callout that deletes the .webpart file from the Web Part Gallery.

**Retracting Schema-based Solutions**

You must take care when retracting schema-based solutions, such as custom list definitions. Many instances can exist, and you probably don't want to break them. Therefore, best practice when instances of these types of solutions are available is to make the Feature invisible to the user rather than retracting the solutions. A small walkthrough in the section on upgrading solutions, later in the article, shows the different steps.

### Removing Windows SharePoint Services Solutions

Windows SharePoint Services solutions that are no longer deployed can be removed from the solution store. You can do this in three ways:

* By using Stsadm.exe
* By using Central Administration
* By using the Windows SharePoint Services 3.0 object model

The following sections describe these methods.

**Using Stsadm.exe**

Administrators can now launch the command-line utility Stsadm.exe, and then execute the **deletesolution** option. The name parameter is the name of the solution and is required. Use the following command to remove a solution.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl39');" \o "Copy Code)

stsadm.exe –o deletesolution –name hellowebpart.wsp

**Using Central Administration**

On the solution management page, which is accessible via the **Operations** tab in Central Administration, you can remove a solution that is no longer deployed. Just click the solution name and use the **Remove Solution** button on the toolbar.

**Using the Windows SharePoint Services 3.0 Object Model**

You can also remove the solution by calling the **Remove** method at the level of the **SPSolutionCollection** object. This collection is exposed via the **SPFarm** class, either for the local farm or from the joined farm. The following code removes a solution from the solution store.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl40');" \o "Copy Code)

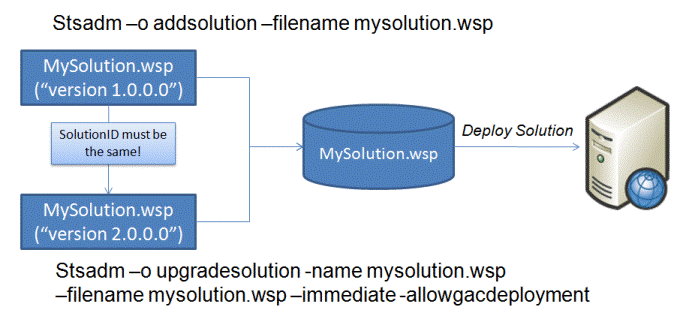
SPFarm.Local.Solutions.Remove("hellowebpart.wsp");

### Upgrading Solutions

The last option for managing SharePoint solutions is to upgrade deployed solutions to new versions. It is important to understand that the versioning is not performed at the level of the Windows SharePoint Services solution. The actual versioning is performed at the level of the solution components (the Feature, the assemblies, and so on).

Figure 25 summarizes how to perform a solution upgrade.

**Figure 25. Upgrading the SharePoint solution**



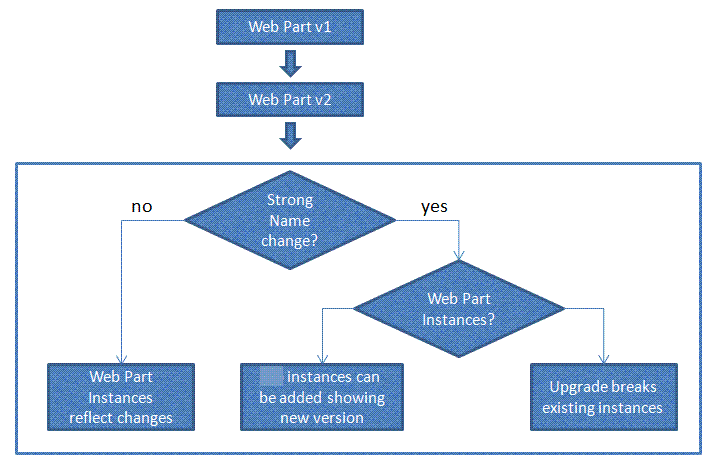
Assume that version 1.0.0.0 of the MySolution.wsp is added to the solution store and deployed to one or more IIS Web applications. A second version of the Windows SharePoint Services solution must have the same SolutionID for an upgrade to succeed. The second version is upgraded by calling the Stsadm.exe command-line utility with the option **upgradesolution**. In the solution store, you provide the name of the solution to upgrade, the new version of the solution, and then specify the options to either schedule the upgrade or have it upgrade immediately. You also specify the options to allow deployment in the global assembly cache and allow for custom code access security policies.

**Guidelines for Upgrading SharePoint Solutions**

When discussing guidelines for upgrading SharePoint solutions, we must make the distinction between code-based solutions (such as Web Parts) and schema-based solutions (such custom list definitions).

* **Upgrading Web Part solutions**   Web Parts are typical examples of code-based solutions. Upgrading a code-based solution often involves replacing the assembly in the private application folder of the IIS Web application or the global assembly cache with an updated version of the assembly. Figure 26 summarizes the possible upgrade paths.

**Figure 26. Upgrading Web Parts**



If you do not change the version number of the .NET assembly, the upgrade process proceeds smoothly. By leaving the version number of the assembly unchanged, you ensure that the metadata stored and made available, for example, in the form of a .webpart registration in the Web Part gallery, does not need to be changed. All the existing instances of the Web Parts on pages upgrade easily.

The upgrade process is a bit more complex when the strong name of the assembly is changed, typically by upgrading the version number from perhaps 1.0.0.0 to 2.0.0.0. If no Web Part instances have been created within the sites, you will not have a problem. The entry in the Web Part gallery is updated with the new .webpart file, and the new version of the assembly is dropped in either the bin folder or the global assembly cache. New additions of the Web Part to the pages reflect the new functionality.

However, users can experience errors when there are existing instances of the older version of the Web Part on the pages, and you decide to perform an upgrade. These instances are still looking for the first version of the assembly. When using the **upgradesolution** option, the older version of the assembly is actually removed, as well as the entry in the **safeControls** section of the web.config file.

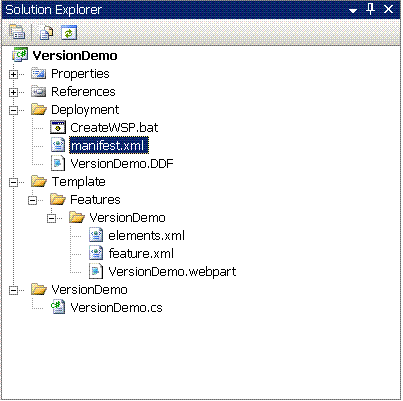
So your approach is a combination of upgrading the Web Part solution with a new version of the assembly combined with two additions in the web.config of the IIS Web application hosting the SharePoint site. These two additions are, first a **SafeControl** element for the first version of the assembly, and second, a **bindingRedirect** ordering ASP.NET to bind to the second version of the assembly, in case there is the request for the first version.

**Walkthrough: Upgrading a Web Part Project**

### To upgrade a Web Part project

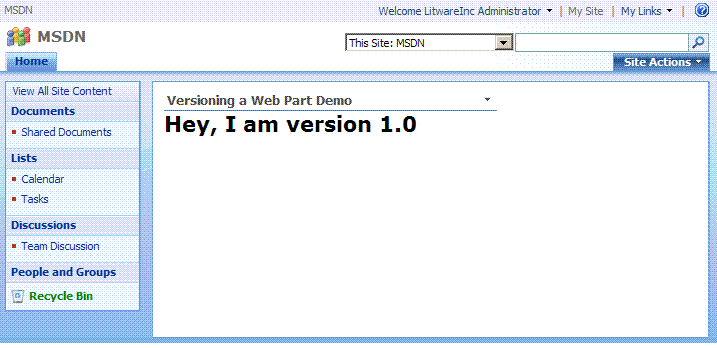
* 1. Start by building a small Web Part project that outputs a string that displays, for example, the version information. You can use Visual Studio 2005 Extensions for Windows SharePoint Services 3.0 to get started, but you must perform the packaging and deployment manually. We advise a project structure such as the one depicted in Figure 27. All of the techniques previously discussed regarding the manual packaging of a SharePoint solution are used here.

**Figure 27. Project structure for a Web Part solution**



* 1. Deploy the solution and activate the Feature in the site collection so that you can add the Web Part to one of your pages (see Figure 28).

**Figure 28. First version of the Web Part**



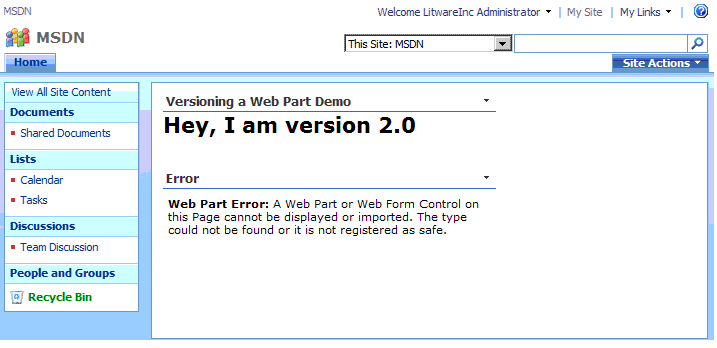
* 1. Assume that there is the request to update the Web Part. Users want to see a slightly different string in the Web Part. Start by changing the rendering of the content in the Web Part and follow up with a change of the version number of the assembly from 1.0.0.0 to 2.0.0.0 using the properties of the Web Part project. Build the project.
  2. Open the .webpart file and update the version number for the **type** element.
  3. Build the project and rebuild the SharePoint solution file. Upgrade the solution by calling the following from either a batch file or a command prompt.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl45');" \o "Copy Code)

Stsadm -o upgradesolution -name VersionDemo.wsp -filename VersionDemo.wsp -immediate -allowgacdeployment

* 1. Refresh the page with the Web Part instance. Figure 29 shows the error displayed as well as a new instance nicely reflecting the change in the output.

**Figure 29. Web Part displaying error and rendering correctly**



* 1. Solve the error that is displayed:
     1. Add the **SafeControl** element for version 1.0.0.0 in the web.config file.

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl47');" \o "Copy Code)

<SafeControl Assembly="VersionDemo, Version=1.0.0.0, Culture=neutral, PublicKeyToken=9f4da00116c38ec5" Namespace="MSDN.Samples" TypeName="VersionDemo" Safe="True" />

* + 1. Add the following element in the web.config file under the **assemblyBinding** element for the redirect operation.

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl48');" \o "Copy Code)

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="VersionDemo" publicKeyToken="9f4da00116c38ec5" culture="neutral" />

<bindingRedirect oldVersion="1.0.0.0" newVersion="2.0.0.0" />

</dependentAssembly>

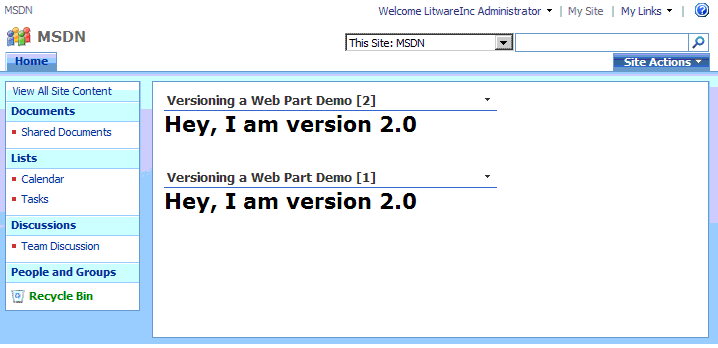
…

</assemblyBinding>

</runtime>

* 1. Refresh the page containing the two Web Part instances. Figure 30 shows the end result. The first version of the Web Part now also works against the 2.0.0.0 version of the assembly.

**Figure 30. Both Web Parts working correctly**



* **Upgrading Schema-Based Solutions**   An example of a schema-based solution is a custom list definition. Generally, to upgrade these types of solutions when instances based on the schema definitions are present on SharePoint sites, you deploy a new version of the Feature and mark the older version as hidden for the user. This way, the instances do not break, and new instances follow the updated definitions.

**Walkthrough: Upgrading Schema-Based Solutions**

Let's return to the Employees list template we created and deployed as a SharePoint solution earlier in the article.

Assume that users have created many instances of this list. The decision has been made to make changes or possibly to completely remove the list definition. As mentioned, you cannot simply remove the solution files from the SharePoint computer. You need to keep them working, although invisible, for the administrators and users. Notice that this approach is also the one to take as an alternative to retracting the solution, as discussed earlier.

Here are the steps to accomplish upgrading a schema-based solution.

### To upgrade a schema-based solution

* 1. Open the folder containing the solution components of the list definition, and then open the feature.xml file.
  2. Change the **Hidden** attribute of the **Feature** element to the value **TRUE**.

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl50');" \o "Copy Code)

<Feature Id="{489C77F1-B064-408e-9B85-029A33BDF9D7}"

Title="Employees"

Description="This feature provides support for creating an Employee List."

Version="1.0.0.0"

Scope="Web"

Hidden="TRUE"

xmlns="http://schemas.microsoft.com/sharepoint/">

<ElementManifests>

<ElementManifest Location="ListTemplates\Employees.xml"/>

<ElementFile Location="Employees\allitems.aspx" />

<ElementFile Location="Employees\dispform.aspx" />

<ElementFile Location="Employees\editform.aspx" />

<ElementFile Location="Employees\newform.aspx" />

<ElementFile Location="Employees\schema.xml" />

</ElementManifests>

</Feature>

This hides the Feature from the administration pages, but you must also hide it on the Create page.

* 1. Open the employees.xml file and add the **Hidden** attribute.

Xml

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl51');" \o "Copy Code)

<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

<ListTemplate

Name="Employees"

Type="10100"

BaseType="0"

OnQuickLaunch="TRUE"

SecurityBits="11"

DisplayName="Employees"

Description="Employees List Type"

Hidden="TRUE"

Image="/\_layouts/images/CHNGCOL.GIF"/>

</Elements>

If you want to retract only the solution, the previous steps are enough.

If, however, you want to replace the list definition with a new version, you’ll need to add the following to the solution package:

* 1. A new version of the Feature (with all of the related files) marked with a different GUID.
  2. An expanded manifest file with the installation steps for both the original (and soon hidden) version, and the new version of the Feature.
  3. An expanded .ddf file containing the entries for packaging the new Feature and the previous version in the .wsp file.

In both scenarios, you must recreate the SharePoint solution file and deploy it by using the **upgradesolution** option for Stsadm, as discussed previously.

[Copy Code](javascript:CopyCode('ctl00_mainContentContainer_ctl52');" \o "Copy Code)

Stsadm.exe -o upgradesolution -filename package\Employees.wsp -name Employees.wsp –immediate

# Conclusion

The concept of solutions in the world of Windows SharePoint Services is extremely important for developers and administrators to understand. Developers who build applications and extend SharePoint sites in numerous ways must package the solution components in SharePoint solution files and deliver those to administrators. Administrators have various options for delivering the solution to the users of the SharePoint sites. This article discusses new options that are available for pushing the solutions from a central solution store to the front-end Web servers and application servers. The article also discusses techniques for maintaining and upgrading the deployed solutions.

# About the Author

Patrick Tisseghem is a Microsoft Office SharePoint Server MVP, and is highly focused on Windows SharePoint Services 3.0 and the Office SharePoint Server 2007. He created and delivered the ISV-focused early adopter material for Microsoft Redmond for the latest version of SharePoint and has toured many countries with his developer-focused workshops. He is a frequent speaker at major Microsoft conferences such as TechEd and SharePoint Connections, and is the author of numerous white papers published on MSDN. He is also the author of a book titled Inside MOSS 2007, published by Microsoft Press. More information about Patrick can be found on his [blog](http://www.u2u.info/Blogs/Patrick) [ http://www.u2u.info/Blogs/Patrick ] .

# Additional Resources

For more information, see the following resources:

* [Development Tools and Techniques for Working with Code in Windows SharePoint Services 3.0 (Part 1 of 2)](http://msdn.microsoft.com/en-us/library/bb530302(printer).aspx) [ http://msdn.microsoft.com/en-us/library/bb530302.aspx ]
* [Patrick Tisseghem blog](http://www.u2u.info/Blogs/Patrick) [ http://www.u2u.info/Blogs/Patrick ]
* [Windows SharePoint Services Developer Center](http://msdn2.microsoft.com/en-us/sharepoint/default.aspx) [ http://msdn2.microsoft.com/en-us/sharepoint/default.aspx ]
* [SharePoint Server Developer Portal](http://msdn2.microsoft.com/en-us/office/aa905503.aspx) [ http://msdn2.microsoft.com/en-us/office/aa905503.aspx ]
* [Microsoft Office Developer Center](http://msdn2.microsoft.com/en-us/office/default.aspx) [ http://msdn2.microsoft.com/en-us/office/default.aspx ]

Tags: wsp wss

http://i.msdn.microsoft.com/Global/Images/wiki.gif

**Community Content**

|  |  |  |
| --- | --- | --- |
| AllowPartiallyTrustedCallers |  | Last Edit 4:11 PM by Medhat Mohamed |

If your Web Part assembly end up in the private \bin folder of the IIS Web application, you can mark your assembly with **AllowPartiallyTrustedCallers** attribute "[assembly: AllowPartiallyTrustedCallers]" instead of using Code Access Security to call an assembly that has a strong name.

Tags:

|  |  |  |
| --- | --- | --- |
| Commonly used Sharepoint CodeSnippets Updated |  | Last Edit 8:46 AM by Adding Custom Layout Pages to WSS 3.0 |

http://sharepointobjectmodel.blogspot.com/

Tags: code sharepoint snippets

Final del formulario