Article • 09/10/2019 • 45 minutes to read

MOSS 2007

Automate Web App Deployment with the SharePoint API

Ethan Wilansky and Paul Olszewski and Rick Sneddon

Code download available at:AutomatingMOSSDeployments2008_05.exe (164 KB)

This article discusses:

- SharePoint deployment basics
- Deploying custom Web Parts
- Customizing site branding
- Deploying lists, forms, and reports

This article uses the following technologies:
SharePoint

Contents

At the 2008 Office System Developer's Conference, one presenter metaphorically described a .NET developer's first look at SharePoint® as equivalent to an experienced mountain climber staring at a smooth wall 100 feet tall and trying to figure out exactly how to scale it. Many Microsoft products offer a dizzying array of approaches to completing a task, and deploying custom SharePoint applications is a good example of

such a case. Because SharePoint is a complex and sophisticated application platform, deployment can present some puzzles to the uninitiated.

In this article, we will show you how to automate custom SharePoint application deployments by taking advantage of the SharePoint API. Following this approach allows you to avoid having to create and maintain a custom site definition. You benefit from a more modular set of components to deploy and manage in your source control system, which is an essential part of a continuous integration effort. Finally, you are following a model that Microsoft will continue to improve in the coming years.

We will explain how you can leverage the SharePoint object model to automate deployment from Web application creation to Feature activation. The Visual Studio® code project included with this article provides a number of code samples that demonstrate how to automate some essential tasks in a custom deployment.

Along the way, we will explore tools and techniques provided by Microsoft and members of the SharePoint developer community that we have found to be especially valuable in your quest toward a fully automated deployment.

SharePoint Deployment Basics

Solutions and Features

The solution packaging framework bundles all the components used to extend WSS into a new file, called a solution file. The solution file is a .cab file format with a .wsp extension and contains a manifest for deploying the various solution parts within the SharePoint framework.

Solutions are added to a SharePoint Web farm and deployed to one or more Web applications. This task can be accomplished with the Stsadm utility or with the SharePoint object model.

Features are the fundamental components used to extend SharePoint solutions. They provide a way of modifying functionality within SharePoint, from adding Web Parts to a gallery to enabling Reporting Services integration.

Features are activated for a single scope--farm, Web application, site collection, or site. The Feature scope is set by the Scope attribute on the Feature element. Features must

be installed before they can be activated, and they must be activated to a scope before they can be used. In addition, Features must be deactivated before they are uninstalled unless they are scoped to the Web application or farm.

An important consideration to bear in mind is how you might scope the automation tasks we have included with this article inside of a Feature. The following figure describes where you might scope the various automation tasks.

From an automated deployment perspective, it is important to know how to programmatically control the state of a Feature. You can use Stsadm to activate and deactivate a Feature, or you can do this programmatically. The following code allows you to enumerate all the Features in a farm and see their display name and scope:

Copy

```
foreach (SPFeatureDefinition definition in
  SPFarm.Local.FeatureDefinitions) {
  Console.WriteLine("Title: {0}, Scope: {1}",
    definition.DisplayName,
    definition.Scope.ToString());
}
```

To activate a Feature programmatically, add the Feature to the Features collection at the level in which the Feature was scoped when it was created, as the following example demonstrates:

Copy

```
SPSite site = new SPSite(https://msdn.fabrikam.com:45001);
SPFeatureDefinition definition =
  SPFarm.Local.FeatureDefinitions["MyFeature"];

if (definition != null &&
  definition.Scope == SPFeatureScope.Site) {
  site.Features.Add(definition.Id, true);
}
```

Scoping for Automation Tasks

Automation	Switch Name	Create	Scope

Task		Solution?	
Create Web app and root Web	CreateWebApp	No	N/A
Web.config management	AddVerifyElement, AddVerifyAttrib, RemoveTrackedItem (in a Feature's FeatureDeactivating event)	Yes	Web application
Apply a theme to site collection	ApplyThemetoSiteCollection	Yes	Site or Web for individual themes
Set master page and css	SetMasterPageandCSS	Yes	Site or Web for individual themes
Exclude items from navigation	ExcludeSiteFromNavigation	Yes	Site or Web
Create a Web	CreatePublishingWeb	Yes	Site
Create a page	CreatePublishingPage	Yes	Web
Add global navigation nodes	AddGlobalNavigationNode	Yes	Web
Add a Web Part to a page	AddListViewWebPart	Yes	Web
Create and modify lists	CreateList, AddItems, CreateView	Yes	Web

Microsoft significantly improved the SharePoint deployment infrastructure with the release of Windows® SharePoint Services (WSS) 3.0 and Microsoft® Office SharePoint Server (MOSS) 2007. The primary components in this deployment infrastructure are the classes in the Microsoft.SharePoint.Administration and Microsoft.SharePoint namespaces and the SharePoint solution deployment framework.

Microsoft also improved the way site definitions and site templates work and how they can be customized. At first glance, it might seem to make sense to use all of these capabilities to support your deployment, but think again. If your goal is to deploy a highly customized portal application in an automated and centralized fashion, you should stay away from site templates and even reconsider a deployment strategy involving custom site definitions.

Most experienced SharePoint developers are likely to agree with our view on site template customization. While they are fine for simple customizations, they are inappropriate for significant customizations because page rendering performance can suffer. Template customizations are stored as differences from the originating site definition and persist as an .stp file in the SharePoint database rather than being served from the file system. When the page renders, the .stp is merged at run time with the underlying site definition in the file system.

What might be more surprising to experienced SharePoint developers is our recommendation to stay away from custom site definitions. Microsoft strongly encourages developers not to customize the out-of-the-box site definitions for a number of reasons. Future service packs may overwrite the existing site definitions and migrations to the next version of SharePoint might not be possible without losing the customizations.

Even if you abide by the recommendation by Microsoft to leave the out-of-the box site definitions alone, there are still more reasons to avoid custom site definitions. For highly customized deployments, site definitions can become huge and unwieldy to manage. A complete site definition can contain multiple Webs. Within each Web there will be one or more pages, and on these pages will be a variety of elements, such as Web Part zones, Web Parts, modules, list definitions, and navigation components. The onet.xml file maintains configuration information about each Web. This file can quickly become a maintenance nightmare as requirements and configurations change. In our experience, it contains between 300 and 600 lines of XML for moderate customizations. If you have five custom Webs in your site definition, suddenly you have between 1,500 and 3,000 lines of XML to maintain.

The SharePoint API and the solution deployment framework provide a better answer. Solutions are deployment packages for SharePoint—they can contain almost anything

you need to deploy to SharePoint. An important part of a solution, though technically not required in all instances, is called a Feature. While the solution is the deployable package, the Feature defines aspects of the package that allow it to employ the SharePoint object model and other managed code and to advertise itself within the SharePoint hierarchy (the farm, Web application, site collection, or Web levels).

Microsoft provides a model for automating the initial installation of SharePoint. Note that the task of configuring a SharePoint instance or farm is typically completed by an administrator. The next step is to create a Web application and site for your custom portal application.

While these tasks can be completed by using SharePoint Central Administration, you have the ability to automate them via the SPWebApplicationBuilder and SPWebApplication classes in the Microsoft.SharePoint.Administration namespace. The SharePoint command-line administrative tool (Stsadm) does not support Web application creation by default. By automating these tasks, they will not only get completed faster, but you can ensure consistency across your single server or farm implementations and as your custom solution moves from development to production.

When SharePoint creates a Web application and generates the top-level site collection and root Web, there is a lot happening behind the scenes. When creating a typical Web application, SharePoint does the following:

- Creates a unique entry in the SharePoint configuration database for the Web app and assigns a GUID to that entry
- Creates and configures a Web application in IIS
- Creates a root folder to store the Web application pages and associated resources
- Creates and configures an IIS application pool
- Configures authentication protocol and encryption settings
- Assigns a Default alternate access mapping for the Web app
- Creates the first content database for the Web application
- Associates a search service with the Web application
- Assigns a name to the Web application that appears in the Web application list in SharePoint Central Administration
- Assigns general settings to the Web application, such as maximum file upload size and default time zone

Optionally, you can add alternate access URLs to the Web application. This is just one of many optional tasks you might want to complete after creating a Web application. And when creating a site collection, SharePoint also creates the top-level site based on a site definition and sets general properties for the site, such as the site title and site owner. Completing these tasks manually can take quite a while—they are tedious and error prone because of the amount of data entry involved. Automating this task speeds up task completion and ensures data consistency from one environment to the next.

As you might expect, there is a fair amount of code involved in automating this task. Rather than show it all here, you should download the code that accompanies this article. You'll find the CreateWebApp method in the SPDeployTests project. (Be sure to read the prerequisites document included in the code sample before attempting to run the console application.) We will emphasize key elements of the code here and call out some specific items that could cause trouble when you automate this part of your custom portal deployment.

One more thing to remember: be sure to run the code from a SharePoint server. It is not designed to run remotely. In our experience, it's best to run an automated SharePoint deployment routine on a SharePoint server in the farm.

Web App Provisioning and Settings

SharePoint Feature and Solution Resources

- Windows SharePoint Services 3.0 SDK
- Office Space: Solution Deployment with SharePoint 2007
- Office Space: Features for SharePoint
- Site Definitions Demystified—Creating a Custom Site Definition Having Custom Web Parts
- Windows SharePoint Services 3.0 Tools: Visual Studio 2005 Extensions
- Implementing Microsoft Office SharePoint Server 2007 and Windows SharePoint Services 3.0 Solutions
- Heather Solomon's blog on branding

In order to create a new Web application, you start by calling the AdministrationService.Farm property from an SPWebService object. Calling the Farm property returns an SPFarm object. The SPFarm object is at the top of the configuration

hierarchy and manages all of the servers, services, and solutions in a SharePoint farm. Once you have the SPFarm object, you can pass it to the SPWebApplicationBuilder constructor:

Copy

```
SPFarm farm =
  SPWebService.AdministrationService.Farm;

SPWebApplicationBuilder webAppBld =
  new SPWebApplicationBuilder(farm);
```

SPWebApplicationBuilder is the worker class for creating SharePoint Web applications. It can complete all of the steps previously outlined for creating a Web application. To make use of this class, set properties of the SPWebApplicationBuilder class, and then call its Create method, which returns an SPWebApplication object for the new Web application:

Copy

```
SPWebApplication webApp = webAppBld.Create();
```

Use the SPWebApplication class to set additional properties, and then call the Update method to send any changes back to the farm. Finally, call the SPWebApplication Provision method to create the IIS Web application and application pool.

Calling these methods in the right order and at the right time in your code is important. Therefore, be sure to review the code samples for this article to see a working example.

When setting properties on a Web application, there are a couple of points to consider. First, be clear on what a setting actually means. If you're not sure what the implications are of a setting, such as choosing Windows NT LAN Manager (NTLM) authentication versus Kerberos, work with a SharePoint infrastructure expert to make this decision. Secondly, some properties can be tricky to set or are unclear as to their affect. For instance, here is an example of setting the ID property in the SPWebApplicationBuilder object:

Copy

```
webAppBld.Id = new Guid("3798E478-4E16-4856-A152-F05EF8C2C777");
```

Like most of the properties in this object, SharePoint will either set the property to a default value or assign a value. In this case, SharePoint will generate a unique GUID and store that along with the Web application in the Objects table of the configuration database. You can run the following query against the SharePoint Configuration database to see all of the Web application objects and their associated ID GUIDs:

Copy

```
SELECT id, name FROM OBJECTS
WHERE Properties
LIKE '%Microsoft.SharePoint.Administration.SPWebApplication%'
```

You can assign this value or let SharePoint assign one for you.

If you are configuring a new application pool and assigning a user account, which is the preferred method of setting up SharePoint, there are a couple of important caveats. First, in a workgroup, ApplicationPoolUserName is the name of a local user, without the server name prepended. If SharePoint is part of an Active Directory® domain, then you should use an Active Directory user account and prepend the domain name for this property:

Copy

```
@"domainname\username"
```

For the ApplicationPoolPassword, you must pass in a SecureString data type for the value. Here is the pattern for building a secure password:

Copy

```
// build the password as a secure string
SecureString appPoolPwd = new SecureString();
appPoolPwd.AppendChar('k');
appPoolPwd.AppendChar('E');
// more of the same to build-up the password
appPoolPwd.MakeReadOnly();
```

Be sure that the user name and password you assign to these values match the values as you've set them in your identity repository before attempting to create the Web application.

The next property, ApplicationPoolId, is simply the name of the application pool as it appears in IIS. It's important to note that the WSS 3.0 SDK help file incorrectly states that ApplicationPoolId is a GUID that identifies the application pool. One could argue that it is a unique identifier within the list of application pools in IIS, but it is neither globally unique nor in the format we typically think of for a GUID.

The next stumbling block is encountered when associating a search service instance with the new Web application. **Figure 1** shows how to accomplish this task and is followed by further explanation as to how you find the proper search service instance.

Figure 1 Associating a Search Service Instance

Copy

```
// get the server hosting the search service
SPServer server = new SPServer("myServer");

// get the office SharePoint server search instances
SPSearchService srchService =
  SPFarm.Local.Services.GetValue<SPSearchService>("SPSearch");

// get the proper instance of the search service
SPSearchServiceInstance searchServiceInst =
  (SPSearchServiceInstance)srchService.Instances[new Guid(
    "C22E9545-71B4-471F-81A7-A213D5A7F8B2")];

// set the search service instance to crawl the web application
webAppBld.SearchServiceInstance = searchServiceInst;
```

There is a service hierarchy in SharePoint where a service contains service instances that actually do the work. Unfortunately, finding a search service instance can be difficult because the instances might have type names, but their Name and possibly DisplayName properties are empty. Therefore, passing the GUID into the Instances collection as in **Figure 1** is one way to set this value.

There are two ways you can find the proper GUID of the search service instance. The first

option is to query the objects table in the SharePoint Configuration database, as demonstrated in the following code:

```
SELECT id, name, properties FROM OBJECTS
  WHERE (Properties LIKE
    '<object type=' +
    '"Microsoft.SharePoint.Search.' +
      'Administration.SPSearchServiceInstance%')
  ORDER by ID
```

Another option is to run the EnumFarmServices method in the code SPDeployTests project contained in the code download. **Figure 2** shows output from one of our servers with the proper search service instance shown in red.

Figure 2 Instances of SPSearch

```
Name: SPSearch
ID: 7fd9c4b0-25c2-4867-b716-cd3b61e7a08e
DisplayName: SPSearch

Instance 1
Name:
ID:c9043683-a431-4cc7-b30a-fa52163f649b
Type name:Windows SharePoint Services Search
DisplayName:

Instance 2
Name:
ID:c22e9545-71b4-471f-81a7-a213d5a7f8b2
Type name:Windows SharePoint Services Search
DisplayName:Search index file on the search server
```

If you want to configure the search service instance from code and you don't select the proper search service instance, Web application creation will fail with a database foreign key dependency error. There are significantly more settings that you can configure for your Web application. The code download provides some examples to get you started.

Creating the first site collection is straightforward. However, before you create the top-

level site collection in a Web farm environment you should reset IIS (using iisreset /noforce). **Figure 3** demonstrates how to create a top-level site collection. For brevity, we moved some of the variables into the input parameters for the Add method. The interesting or potentially confusing variables are declared and described above the Add method.

Figure 3 Creating a Site Collection

Copy

```
// for the top-level site
string url = "/";

// this is the MS language Locale ID.
// see https://www.microsoft.com/globaldev/reference/lcid-all.mspx
uint LCID = 1033;

// this is the site template for the
// Publishing Site - Collaboration Portal merged site definition
string template = "SPSPORTAL";

// create a site collection with an out of the box site definition
  SPSite MSDNSiteCollection = webApp.Sites.Add(url,
  "The MSDN site collection", "a description of this site",
  LCID, template, "mydomain\\myLoginName",
  "my owner name", "my email address");

// close the site collection
MSDNSiteCollection.Close();
```

Managing Web.config

A common task in most SharePoint code projects is updating web.config for each Web application. While updates can be made manually in small implementations, that doesn't scale to large, multi-developer projects. Keeping a copy of a SharePoint Web application's web.config in a source control system and having developers check out the file, update it for their project, and check it in is one obvious possibility. You could then script the web.config deployment to your Web front-end servers on a schedule or based on a particular event, such as check-in. However, this approach does not take advantage of the ability of SharePoint to manage web.config settings for you.

If you use the SharePoint API to update web.config, new front ends added to the SharePoint farm receive the updates to the web.config and SharePoint maintains these settings in existing front ends. There are two common approaches for getting SharePoint managed updates into web.config programmatically. For Web Part SafeControl entries and custom access control policies, rely on the SharePoint solution packaging framework. For unique entries, use the SPWebConfigModification class in the Microsoft.SharePoint.Administration namespace.

Figure 4 shows how to add an XML child element to your web.config file using SPWebConfigModificationType.EnsureChildNode. In this example, the SPWebConfigModification class ensures that the <add key = "key01" value = "Setting01" /> element appears inside of the appSettings element. The comments in the code explain how to achieve this result. Notice the single quotes for the values 'key01' and 'Setting01' in the xmlValue variable. Using single quotes makes the code clearer and the SPWebConfigModification class takes care of changing these to double quotes when the values are written to each front-end Web server.

Figure 4 Adding a Child Node in Web.config

Copy

```
// get the webApp
SPWebApplication webApp =
  new SPSite("https://msdn.fabrikam.com").WebApplication;

// the element to add or verify contains
// the following name and attributes:
string elementName = "add[@key='key01'][@value='Setting01']";

// the location where the new element should be added or verified is
// present in the target web.config
string xPath = "configuration/appSettings";

// the value exactly as it should be written
string xmlValue = @"<add key = 'Key01' value = 'Setting01' />";

SPWebConfigModification modification =
  new SPWebConfigModification(elementName, xPath);

// this name determines who can modify this entry –
// only the owning name (typically a fully qualified program name,
// GUID or possibly the WebConfigModificationFeatureReceiver.OwnerID
```

```
// name can modify the entry
modification.Owner = "SPWebConfigTestAddChild";

// for identical entries, this controls what order the entry is made
// typically fine to leave this at 0
modification.Sequence = 0;

// EnsureChildNode means to add or verify that the child node is
present
// as specified by the Value property next
modification.Type =
  SPWebConfigModification.SPWebConfigModificationType.EnsureChild-
Node;

// the actual value to write
modification.Value = xmlValue;

// add the modification to the SPWebConfigModification collection
webApp.WebConfigModifications.Add(modification);

// check all web applications on the farm front ends to ensure
// that the web.config modification(s) have been applied
webApp.Farm.Services.GetValue<SPWebService>().ApplyWebConfigModifica-
tions();

// propagate changes in web application across the farm
webApp.Update();
```

You can also use the SPWebConfigModification class to add, change, or ensure that an attribute value within an XML node appears in web.config. This is simple when you're writing or verifying an XML attribute that is unique in the web.config file. For example, you might want to change the autoDetect attribute from true to false in the following unique XML fragment:

Copy

```
<system.net>
  <defaultProxy>
    <proxy autoDetect="true" />
  </defaultProxy>
</system.net>
```

In this case, the SPWebConfigModification constructor looks like the following:

```
SPWebConfigModification modification =
  new SPWebConfigModification(
  "autoDetect", "system.net/defaultProxy");
```

The SPWebConfigModificationType enumeration is then set to EnsureAttribute, like this:

```
modification.Type =

SPWebConfigModification.SPWebConfigModificationType.EnsureAttribute;
```

EnsureChildNode and EnsureAttribute are the two values in this enumeration that you should use for your updates. There are a number of blog entries suggesting that EnsureSection (the third enumeration) works for writing whole sections into web.config. Unfortunately, you cannot remove the entries easily.

Other than the differences mentioned, there is little difference to the pattern you use for writing an attribute versus writing one or more XML nodes to web.config. What's a little trickier is writing or verifying an attribute when there are identically named elements inside of a parent element (for example, multiple add child elements in the appSettings element). In this case, the xPath string gets you to the correct element.

Consider the following appSettings element:

```
<appSettings>
  <add key="FeedCacheTime" value="300" />
  <add key="FeedPageUrl"
    value="/_layouts/feed.aspx?" />
  <add key="FeedXsl1"
    value="/Style Library/Xsl Style Sheets/Rss.xsl" />
  <add key="key01" value="Setting01" />
</appSettings>
```

In order to change the value attribute of the fourth add element, use an xPath statement that specifies the child element, like so:

```
configuration/appSettings/add[4]
```

xPath syntax is the key to pinpointing exactly what you need to update. One reference for xPath syntax is available at msdn2.microsoft.com/ms256115 .

Figure 5 shows how to modify or verify that the value attribute is set to Setting02 in the fourth "add" child element of the appSettings element. For brevity, we removed comments that appear in **Figure 4**. Now that these settings have been added, you can inspect the web.config file on any Web front end running the target Web application.

Figure 5 Modifying the Fourth Attribute of a Child Node

```
SPWebApplication webApp =
  new SPSite("https://msdn.fabrikam.com").WebApplication;

// the attribute's name to which a value will be added,
// changed or verified:
string attributeName = "value";

// the location of the element that should be modified or verified
string xPath = "configuration/appSettings/add[4]";

// the value of the attribute as it should be written
string attributeValue = "Setting02";

SPWebConfigModification modification =
  new SPWebConfigModification(attributeName, xPath);

modification.Owner = "SPWebConfigTestModifyAttribute";

modification.Sequence = 0;

// EnsureAttribute means to add or verify that an attribute value is
// present as specified by the Value property next
modification.Type =
  SPWebConfigModification.SPWebConfigModificationType.EnsureAt-
tribute;

modification.Value = attributeValue;
```

```
webApp.WebConfigModifications.Add(modification);

webApp.Farm.Services.GetValue<SPWebService>().ApplyWebConfigModifica-
tions();

webApp.Update();
```

Figure 6 shows how to see what settings are being managed by SharePoint by enumerating the collection of settings contained in the WebConfigModfications property of the Web app. Manually removing or changing the entries in web.config that are managed by SharePoint won't do you any good in the long term. SharePoint tracks these entries and ensures that they are in web.config each time the WSS service restarts.

Figure 6 Enumerating a List of Modifications

Copy

```
SPWebApplication webApp =
 new SPSite("https://msdn.fabrikam.com").WebApplication;

// use the System.Collection collection class to get
// a collection of modifications in the configuration database
Collection<SPWebConfigModification> collection =
  webApp.WebConfigModifications;

// iterate the collection of modifications and return properties
// about the items
foreach (SPWebConfigModification item in collection)
{
 Console.WriteLine("\nitem {0}", collection.IndexOf(item));
 Console.WriteLine("owner is: {0}", item.Owner);
 Console.WriteLine("path is: {0}", item.Path);
 Console.WriteLine("name is: {0}", item.Name);
 Console.WriteLine("value is: {0}", item.Value);
 Console.WriteLine("sequence is: {0}", item.Sequence);
}
```

To remove entries, you need to know the value of the Owner property for the modification. Then you can use the Remove method of the SPWebApplication's WebConfigModificatons property to delete entries maintained by SharePoint. See Vincent Rothwell's blog post at blog.thekid.me.uk/archive/2007/03/20/removing-web-config-entries-from-sharepoint-using-spwebconfigmodification.aspx for more

information about removing entries. There is also a sample in the code download for this column showing how to remove an entry.

Modifying web.config can be tricky using the SPWebConfigModification class. There are a number of caveats that you should be aware of. You can read about the most common ones at Reza Alirezaei's blog (blogs.devhorizon.com/reza/?p=459) and Mark Wagner's blog (www.crsw.com/mark/default.aspx). We highly recommend you take a close look at these articles before getting started using this class. After you've done your research, be sure to carefully test your modifications and your ability to remove these modifications using the SPWebConfigModification class in an isolated environment, in a shared development environment, and in a Web farm development environment.

Even with the ability of SharePoint to manage web.config settings, there are instances when you might want to manage settings without the SharePoint API, for example, managing a unique section of sensitive credential information. In this case, consider using the WebConfigurationManager in the System.Web.Configuration namespace. To encrypt the information, you can use either the aspnet_regiis utility or use the AppSettingsSection class in the System.Configuration namespace. You can read about how to edit and encrypt Web.Config sections using C# 2.0 at c-sharpcorner.com/UploadFile/neo_matrix/EditWebConfig05042007091116AM/EditWebConfig.aspx.

Custom Web Parts

Microsoft built the solution packaging and deployment framework with Web Parts in mind. The key to end-to-end Web Part deployment starts with the Web Part deployment file, officially known as the Web Part Control Description file. There are two Web Part deployment formats: .WebPart and .dwp. Both are XML files with underlying schemas. Web Parts Control Description Files at msdn2.microsoft.com/library/ms227561(VS.80).aspx describes the .WebPart schema.

The .dwp format is available for backward compatibility. It was the SharePoint Web Part control description file format in WSS 2.0 and SharePoint Portal Server 2003. While you can use this format for your Web Part deployment in WSS 3.0 and MOSS 2007, you should avoid .dwp if your goal is to fully automate your Web Part deployments. This is because the .dwp schema does not include some of the essential settings that are required for full automation. For example, if you need to hide the chrome (otherwise

known as the styling) of a custom Web Part, you can't do that directly with a .dwp file.

How you begin creating your custom Web Part affects the type of deployment file that is generated and is expected for deploying your component. Surprisingly, there is little guidance explaining the scenarios that automatically generate a .dwp file or a .WebPart file. If you deploy a Web Part to SharePoint using the older .dwp format, the only way we have found to switch to a .WebPart file is to uninstall the solution or, if you're not using a solution, manually remove the Web Part from SharePoint, rebuild the project, and recreate the solution with a .WebPart file.

There are two approaches you can take to ensure that your custom Web Part is deployable using the newer ASP.NET .WebPart format. The first approach is to start a Visual Studio class project. Note that when you declare your Web Part class, instead of inheriting from the SharePoint Web Part class (Microsoft.SharePoint.WebPartPages.WebPart), inherit from the ASP.NET Web Part class, like so:

Copy

```
public class WebPart1 : System.Web.UI.WebControls.WebParts.WebPart
```

Microsoft recommends inheriting from the ASP.NET Web Part class in most of its documentation chiefly because a Web Part that inherits from this class is potentially deployable both to SharePoint and to a custom ASP.NET portal. We say potentially because a Web Part that interacts with the SharePoint object model might be dependent on running in the context of a SharePoint portal. If you end up requiring one of the handful of features available in the SharePoint Web Part class, you can later switch to inheriting from the SharePoint namespace. However, you should create your SharePoint solution and your .WebPart file before making this switch.

The second approach to ensuring that a .WebPart deployment file is automatically created is by using the Visual Studio Extensions for WSS 3.0 or the new WSPBuilder Extension. When using these tools and the included SharePoint Web Part project template, it doesn't matter which namespace you use when inheriting from the Web Part class.

Once you have a .WebPart file, the next step in full deployment automation has more to

do with the solution, which we explore only briefly in this article. For now, the key is to get the .WebPart file prepared for the deployment.

While you can manually add elements to your .WebPart file, the following approach will make this effort significantly easier. In the constructor for your Web Part, add the following code:

```
// make the web part deployment file exportable
this.ExportMode = WebPartExportMode.All;
```

When you build and later install and configure your Web Part, this allows you to export the .WebPart file. You can also manually set this in the tool part properties instead of declaring this in code.

Next, install your Web Part into SharePoint or a custom ASP.NET Web application and drop it onto a Web Part zone on the page. (Later in this article, we will describe ways to automate the initial Web Part deployment.) From the Web Part's Tool Part Properties dialog, configure the Web Part exactly as you would like it to appear and function on the page. From the edit menu of the Web Part, click Export. Save the exported .WebPart file into the Visual Studio project created for the Web Part.

Figure 7 shows a basic .WebPart file as prepared by version 1.1 of the Visual Studio Extensions for WSS following the configuration steps outlined previously. The settings for custom toolpart properties and custom toolparts with properties will also appear in the .WebPart export after configuring their settings.

Figure 7 Web Part XML File

Prior to Configuration

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
  <webPart xmlns="https://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <!--
        The following Guid is used as a reference to the web part
```

```
class,
        and it will be automatically replaced with actual type name
        at deployment time.
      -->
      <type name="5ad2f5ec-8c67-4ca9-b2fb-83c4dadd50c0" />
      <importErrorMessage>
        Cannot import WebPart1 Web Part.</importErrorMessage>
    </metaData>
    <data>
      <properties>
        <property name="Title" type="string">WebPart1 Web Part</prop-
erty>
        <property name="Description" type="string">
          WebPart1 Description</property>
      </properties>
    </data>
  </webPart>
</webParts>
```

After Configuration

Copy

```
<webParts>
  <webPart xmlns="https://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <type name="WPUsingVSeWSS1dot1.WebPart1, WPUsingVSeWSS1dot1,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=9f4da00116c38ec5" />
      <importErrorMessage>
        Cannot import WebPart1 Web Part.</importErrorMessage>
    </metaData>
    <data>
      <properties>
        <property name="AllowClose" type="bool">False</property>
        <property name="Width" type="unit">170px</property>
        <property name="AllowMinimize" type="bool">False</property>
        <property name="AllowConnect" type="bool">True</property>
        <property name="ChromeType" type="chrometype">None</property>
        <property name="TitleIconImageUrl" type="string" />
        <property name="Description" type="string">
          WebPart1 Description</property>
        <property name="Hidden" type="bool">False</property>
        <property name="TitleUrl" type="string" />
        <property name="AllowEdit" type="bool">True</property>
        <property name="Height" type="unit">25px</property>
        <property name="HelpUrl" type="string">
          /pages/SimpleHelp.htm</property>
```

```
        <property name="Title" type="string">WebPart1 Web Part</prop-
erty>
        <property name="CatalogIconImageUrl" type="string" />
        <property name="Direction" type="direction">NotSet</property>
        <property name="ChromeState" type="chromestate">Normal</prop-
erty>
        <property name="AllowZoneChange" type="bool">False</property>
        <property name="AllowHide" type="bool">False</property>
        <property name="HelpMode" type="helpmode">Modal</property>
        <property name="ExportMode" type="exportmode">All</property>
      </properties>
    </data>
  </webPart>
</webParts>
```

Unfortunately, if your Web Part uses the connection provider or connection consumer interface, you have to either programmatically wire up your Web Parts or, more commonly, allow the connection to be made at run time via the Web Part edit pulldown menu. The tool part connection settings aren't saved in the export file. If you don't want to wire up your Web Parts in the Web Part code, use the SharePoint object model to automate this part of the deployment. **Figure 8** shows how to programmatically connect the provider and consumer connection points of two Web Parts.

Figure 8 Connecting Web Parts

Copy

```
// webPartPage is an SPFile object
// page check out is required before modifying publishing pages
webPartPage.CheckOut();

// get the web part manager for the page
SPLimitedWebPartManager webPartMgr =
  webPartPage.GetLimitedWebPartManager(PersonalizationScope.Shared);

// providerWebPart and consumerWebPart are of type
// System.Web.UI.WebControls.WebParts.WebPart

// providerConnection is type
// System.Web.UI.WebControls.WebParts.ProviderConnectionPoint

// consumerConnection is type
// System.Web.UI.WebControls.WebParts.ConsumerConnectionPoint
```

```
// connect the web parts
  webPartMgr.SPConnectWebParts(providerWebPart,
  providerConnection,   consumerWebPart, consumerConnection);

// update, check in, and publish the page
webPartPage.Update();
webPartPage.CheckIn("");
webPartPage.Publish("");
```

You obtain the consumer and provider connection points for the corresponding Web Parts from the collections returned by the GetConsumerConnectionPoints and GetProviderConnectionPoints methods of SPLimitedWebPartManager. This approach works if the connection types are compatible. If they are not, such as when you need to pass filter values from a QueryStringFilter Web Part to a consumer Web Part, you add a transformer to the SPConnectWebParts call, like so:

Copy

```
TransformableFilterValuesToFilterValuesTransformer
  transformer = new
    TransformableFilterValuesToFilterValuesTransformer();

webPartMgr.SPConnectWebParts(providerWebPart,
  providerConnection, consumerWebPart,
  consumerConnection, transformer);
```

Simplified Site Branding

We have already discussed creating a Web application and site collection. We can now turn our attention to modifying the default site appearance. There are many different ways to brand a MOSS 2007 site, but we will address one approach to provisioning and branding that builds on the out-of-the-box SharePoint collaboration portal, publishing pages, a custom master page, style sheets, page layouts, and themes. While the focus is on publishing sites, keep in mind that many of these techniques are applicable to non-publishing sites as well.

Let's start by saying the following: do not directly modify any of the out-of-the-box SharePoint components. Always make a copy and move it to a separate work space and use your favorite code editor for this task.

SharePoint master pages are implemented in the same way as any ASP.NET 2.0 master page. Once you've created the master page in your favorite tool, upload it to a virtual folder called /_catalogs/masterpage within the top-level SharePoint site. You can then apply the master page to an individual page, site, or entire site collection using the SPWeb class in the Microsoft.SharePoint namespace. MasterUrl and CustomMasterUrl are the two properties of the SPWeb class that identify site master pages. These two properties refer to ~masterurl/default.master and ~masterurl/custom.master, respectively, in the MasterPageFile attribute of the page layout. Setting MasterUrl changes the system master page for forms and views within the site, and setting CustomMasterUrl affects publishing pages within the site. As you might imagine, changing the system master pages does not affect pages below /_layouts in the address, such as the Administrative Site Settings pages and Access Denied page. These inherit from the application.master or simple.master pages.

In MOSS 2007, you can use the AlternateCssUrl property of the SPWeb class to specify an overriding style sheet for a site. WSS sites do not support this option, so your alternative in that environment is to specify the CSS link in the master or content pages. A good location for the CSS file, if it is to be maintained on the site, is in /Style Library. However, any location that is accessible via a URL path in SharePoint is fine, too.

The next step is to apply a theme. A theme presents a way to skin the SharePoint user interface with new colors, styles, and images using CSS. Themes are stored in the \Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE\THEMES folder on the MOSS server.

A theme is applied at the site scope and is not inherited by child Webs within the site collection. Because of their scoping, themes modify the appearance of all pages within the root Web, including the Site Settings pages. This code sets the Theme for all Webs within a site collection:

Copy

```
// iterate all sites in the site collection
foreach (SPWeb webSite in MSDNSiteCollection.AllWebs)
{
  // apply the Lacquer theme
  webSite.ApplyTheme("Lacquer");
```

```
  // update the site
  webSite.Update();
}
```

You can extend this example to also set the master page and style sheet for Webs that do not inherit settings from the parent site. By default, our approach causes all new child Webs to have the same style settings as the top-level Web.

Creating customized page layouts for your publishing content pages is another option for site branding and goes hand-in-hand with master pages. For information on branding SharePoint, be sure to review Heather Solomon's blog posting on branding starting at the landing page named at heathersolomon.com/blog/articles/sp2007.aspx.

Once you have finished creating your page layouts, you must upload them to the /_catalogs/masterpage gallery, along with the master page. **Figure 9** shows how to create a publishing page by selecting a desired page layout from the list of available layouts for the publishing Web site.

Figure 9 Creating a New Page from a Page Layout

Copy

```
PageLayout myLayout = null;

// open a web site
SPWeb webSite = MSDNSiteCollection.OpenWeb();

if (PublishingWeb.IsPublishingWeb(webSite)) {
  // get the publishing web instance for this site
  PublishingWeb publishingWeb = PublishingWeb.GetPublishingWeb(web-
Site);

  // get the collection of publishing pages
  PublishingPageCollection ppages = publishingWeb.GetPublishing-
Pages();

  // iterate the available page layouts until we find the one we want
  PageLayout[] layouts = publishingWeb.GetAvailablePageLayouts();
  for (int index = 0; index < layouts.Length; index++) {
    if (layouts[index].Name.Equals("my_page_layout.aspx",
      StringComparison.OrdinalIgnoreCase)) {
      myLayout = layouts[index];
```

```
      break;
    }
   }

  // add the page to the publishing pages collection
  newPage = ppages.Add("sample_page.aspx", myLayout);

  // set the page title
  newPage.Title = "Test Page";

  // update the page, check in and publish
  newPage.Update();
  newPage.CheckIn("");
  newPage.ListItem.File.Publish("");

  webSite.Update();
}
```

Modifying navigation is another common branding task since it affects what users can see and how they can proceed through a site hierarchy. The Microsoft.SharePoint.Publishing namespace exposes several classes that target the Publishing site infrastructure, such as PublishingWeb and PublishingPage. Using these classes, we can easily modify navigation for each site.

If you want a child Web to display as a root level site in global navigation, first turn off inheritance from the parent site, like so:

Copy

```
publishingWeb.InheritGlobalNavigation = false;
```

You might also want to hide all site pages from global navigation. Setting IncludePagesInNavigation to false hides all pages in the site, regardless of whether the PublishingPage.IncludeInGlobalNavigation property is set to true:

Copy

```
// do not show pages in navigation
publishingWeb.IncludePagesInNavigation = false;
```

If you are dealing with default sites that don't inherit from PublishingWeb, it's still

possible to hide these sites from the global navigation bar. For example, if you create a site collection using the collaboration portal template and want to exclude the News site from global navigation, add that site to the __GlobalNavigationExcludes property of the site:

Copy

```
string globalNavExcludes = String.Empty;
SPWeb webSite = MSDNSiteCollection.RootWeb;
// _GlobalNavigationExcludes property contains a delimited string of
// GUIDs identifying the Id of each site to be excluded from global
// navigation

if (webSite.AllProperties.ContainsKey("__GlobalNavigationExcludes"))
{
  globalNavExcludes =
    webSite.AllProperties["__GlobalNavigationExcludes"].ToString();
}

SPWeb newsSite = MSDNSiteCollection.AllWebs["News"];
// string is delimited "{GUID};{GUID};",
// use format code B to convert to string
globalNavExcludes += String.Concat(currentWeb.ID.ToString("B"), ";");

webSite.AllProperties["__GlobalNavigationExcludes"] = globalNavEx-
cludes;
webSite.Update();
```

Adding navigation nodes directly to an SPNavigationNodeCollection is a good way to display only the nodes you want as well as to group nodes and links to external sites. **Figure 10** shows how to add an internal link, external link, and a heading to the global navigation bar. This example addresses some of the properties of the SPNavigation class that affect whether the link opens in a new window and how to handle empty URLs.

Figure 10 Creating Navigation Nodes

Copy

```
SPWeb webSite = MSDNSiteCollection.OpenWeb("test", false);

if (PublishingWeb.IsPublishingWeb(webSite)) {
  // get the publishing web instance for this site
  PublishingWeb publishingWeb = PublishingWeb.GetPublishingWeb(web-
```

```
Site);

  // get the collection of global navigation nodes
  SPNavigationNodeCollection navNodes =
    publishingWeb.GlobalNavigationNodes;

  // add an internal link as the first node
  SPNavigationNode internalLink = new SPNavigationNode("Help",
    "Pages/Help.aspx", false);
  navNodes.AddAsFirst(internalLink);

  // add an external link after the internal link node
  SPNavigationNode externalLink = new SPNavigationNode("MSDN",
    "https://msdn2.microsoft.com/en-us/default.aspx", true);
  navNodes.AddAsLast(externalLink, internalLink);

  // set the node to open in a new window
  navNode.Properties["Target"] = "_blank";
  navNode.Update();
  // add a header node with some children
  SPNavigationNode headerNode = new SPNavigationNode(
    "Header", "", false);
  navNodes.AddAsLast(headerNode);

  // set the node to be a header element
  headerNode.Properties["UrlFragment"] = "";
  headerNode.Properties["NodeType"] = "Heading";
  headerNode.Properties["BlankUrl"] = "True";
  headerNode.Update();

  // create the child nodes
  SPNavigationNode child1Node = new SPNavigationNode("Test1",
    "Pages/Test1.aspx", false);
  headerNode.Children.AddAsLast(child1Node);

  SPNavigationNode child2Node = new SPNavigationNode("Test2",
    "Pages/Test2.aspx", false);
  headerNode.Children.AddAsLast(child2Node);

  headerNode.Update();
  publishingWeb.Update();
}
```

There is a lot more to branding a site than what we've covered here. However, these code samples should help you get started with deploying customizations and configuring a site collection to apply your brand.

Setting Up Lists

Because lists are a fundamental feature of any SharePoint site, getting them deployed in an automated way is important. There are two common approaches for automating list deployment: deploying list definitions and deploying individual populated list instances. The approach you take really depends on exactly what you want to achieve. If you want list structures that can be created from list templates, then list definitions are the way to go. If your goal is to deploy a single instance of a list with values, then use the object model to generate a deployable list instance.

If you decide list definitions are what you need, you can use the SharePoint Solution Generator utility in Visual Studio Extensions for WSS to create a list definition Visual Studio project. Using the generator, you choose a Web application or specify a Web application, as shown in **Figure 11**.

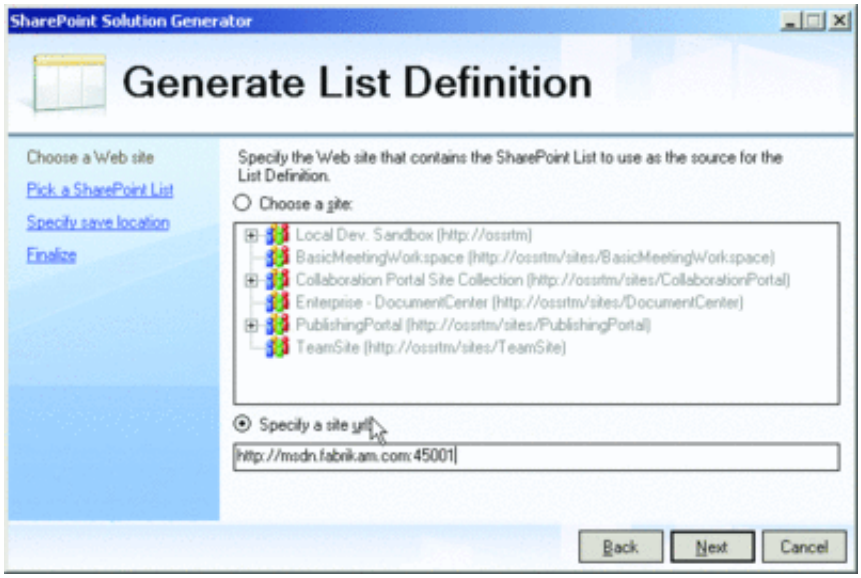

Figure 11** Generate List Definition Wizard **(Click the image for a larger view)

The wizard then has you select a target site collection or a specific Web within a site collection as the target of the extraction process. If you choose the List Definition option in Visual Studio Extensions for WSS, you can pick specific lists to include in the Visual Studio project. Before extracting your lists to a project, be sure that you've configured the source lists exactly as you want them to appear when they are deployed elsewhere.

If you're creating list instances in your custom deployment that you don't want templated, you can use the object model to create lists and configure list settings, such

as adding a view. Note that the key classes for this operation are SPListCollection, SPList, SPListItemCollection, SPListItem, and SPViewCollection in the Microsoft.SharePoint namespace. The WSS 3.0 SDK does a good job of covering these classes.

The pattern for creating a list, creating items in a list, and creating a view all involve getting a collection, creating an object, and adding the object to the collection. For creating a list and adding items to it, you also must call the Update method to submit the update to the server. Review the SPListConfig class in the code download for simple methods that demonstrate how to complete all of these tasks with code. Note, though, that to keep the code download as simple as possible, we have left out object disposal and error handling.

The trickiest part of the effort will be creating the view in your code because programmatic view configuration is achieved using the Collaborative Application Markup Language (CAML) XML query syntax. One way to simplify the process is to create a view for a list through the SharePoint Web interface, and then extract your list using the SharePoint Solution Generator. Finally, open the schema.xml file and find the View tag containing the name of your custom view. Within that tag, look for the Query element and copy the XML within it to a string variable in your code. This is the CAML that gets passed in as the third parameter of the Add method in the SPViewCollection class (views object), as shown here:

Copy

```
StringCollection fieldsInView = new StringCollection();

string query =
  "<GroupBy Collapse=\"TRUE\" GroupLimit=\"100\">" +
    "<FieldRef Name=\"" + _fieldBool + "\" />" +
  "</GroupBy>" +
  "<OrderBy>" +
    "<FieldRef Name=\"Title\" />" +
  "</OrderBy>";

views.Add("myView", fieldsInView, query, 10, true, true);
```

You can also use the Lists class in the Lists Web service to complete these tasks. However, for an automated deployment, working with the object model directly is faster.

Form and Report Deployment

InfoPath® form development is a large topic unto itself, and there are many resources to help you understand how to develop InfoPath form applications. Using the InfoPath form templates from Visual Studio 2008 or Visual Studio 2005 is our preferred approach.

Once you have your InfoPath form application completed, the next step is to publish it to either a SharePoint farm or to a network location from Visual Studio. When you click Publish on the Build menu in Visual Studio, the Publishing Wizard deploys an .xsn file to the target location, which is an InfoPath form package containing your form project components, such as an .xsf InfoPath form, an assembly, and various XML and schema files. The process of adding the InfoPath package into the Central Administration—Application Management—Manage Form Templates library causes SharePoint to build a solution file (.wsp package) for you, which is automatically added to the target SharePoint farm.

We haven't given any detail on SharePoint solutions or methods for building them up to this point. For the purpose of automating deployment of your InfoPath form application, the key is to recognize that SharePoint builds the solution file and adds it to the farm for you. After that's accomplished, you can use Mark Wagner's simple-but-powerful solutionExport tool to extract the .wsp file for deployment to other SharePoint farms. You can read about this tool and download it from Mark Wagner's blog (mentioned previously).

There are two very important caveats to keep in mind here. The first is that the solution is only updated when you deploy a new .xsn file to the farm. So if you are planning to deploy the solution package to a different farm environment, you need to make sure that the data connection used by your InfoPath form points to the correctly configured data connections. In addition, the .udcx data connection files do not get packaged with the InfoPath form .xsn package or the resulting .wsp solution package.

One way to automate the deployment of the data connection files is to create another solution package that contains a Feature to deploy the .udcx files to a target data connection library when the Feature is activated. As you move your InfoPath Solution files from one SharePoint instance to the next, you will have a much easier time deploying them if you keep the relative path to your form library and data connection library the same. This is a perfect example of convention over configuration.

With the release of SQL Server 2005 SP2, Microsoft created a new mode in SQL Server Reporting Services that integrates reports into SharePoint. This new mode is called SharePoint Integrated Mode. Unfortunately, Microsoft didn't address how to automate the deployment of reports and their associated data connections into SharePoint.

Fortunately, Shawn Feldman has come to the rescue with his brilliant blog entry, Automating Report Deployment with Reporting Services in SharePoint Integration Mode, at feldrox.spaces.live.com/blog/cns!C46024CE04ED4278!469.entry. In this blog, he explains exactly how you can automate report deployment. In fact, this entry explains related tasks and how to automate them using the SharePoint object model and the ReportingServices2006 Web service. Here are the tasks he demonstrates using these APIs:

- Creating a reports library in SharePoint
- Adding a Content Type to the library
- Creating a data source and adding it to a SharePoint data connection library
- Adding a Reporting Services report to the reports library
- Associating a Reporting Services report with the data source

Shawn doesn't cover programmatic Feature activation; however, we demonstrate how you can accomplish this step in the "Solutions and Features" sidebar.

There are many other deployment tasks we couldn't cover here, such as configuring SharePoint permissions, integrating unique IIS Web applications that run in the context of a SharePoint instance, and deploying HTTP handlers and updating web.config following their deployment. The tasks we did cover, however, should get you well on your way to fully automating your deployment.

What Next?

We have given you a whirlwind tour of how to automate the most common tasks involved in getting a custom SharePoint application deployed. The next steps to further your automation efforts are to continue your exploration of the SharePoint object model, work with one or more solution packaging tools, and move appropriate automation steps into your solution packaging framework.

The "SharePoint Feature and Solution Resources" sidebar provides links to some

additional helpful articles and blog posts on developing and deploying elements of your SharePoint application. In addition, we've provided more information about creating solutions in the "Some Approaches for Creating Solutions" sidebar.

Some Approaches for Creating Solutions

While you can create solutions manually, there are a number of tools released by both Microsoft and the SharePoint community to ease the pain of building and deploying such solutions.

Visual Studio Extensions for WSS 3.0

Visual Studio Extensions for WSS 3.0 is an effective tool for automated SharePoint component build and deployment activities and it's fully integrated into Visual Studio 2005. A distinct advantage of the Extensions for WSS is the deployment capability it adds to Visual Studio. But while this deployment capability works really well for individual solutions, it is not ideal for large teams where full control and automation of a centralized deployment is important.

The biggest limitation with the Extensions for WSS is that it is designed to run in a Visual Studio installation on a SharePoint server. You can get the extensions installed on Windows XP or Windows Vista and use a limited set of capabilities, such as creating a Web Part project, by adding the following registry entry:

Copy

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\Web Server Exten-
sions\12.0]
"SharePoint"="Installed"
```

Other developers who want to work on your code must have the Extensions for WSS installed or they won't be able to open your project.

Some encouraging developments with Visual Studio Extensions for WSS version 1.1 include the ability to edit your .wsp solution and build it in real-time in the WSPView pane. This provides a powerful way to work with the files and structure of your solution package. In addition, when you deploy a solution, a copy of the .wsp file is dropped into the build folder for your project so that you can deploy it independently of the

Extensions for WSS deployment mechanism.

Visual Studio Extensions for WSS version 1.2, which is planned to support Visual Studio 2008, is expected to ship this year. Read about the new version on the Microsoft SharePoint Products and Technologies Team Blog at blogs.msdn.com/sharepoint/archive/2008/02/11/announcing-the-final-release-of-vsewss-1-1-and-the-upcoming-version-1-2.aspx.

WSP Builder

WSP Builder, by Carsten Keutmann, is an outstanding tool worth considering for your SharePoint solution build efforts. WSP Builder is particularly useful for continuous integration scenarios and in situations where developers hand off the packaging effort.

The tool creates solution files based on a folder structure that mimics the SharePoint \12 folder, as well as a few other specific folders. In Visual Studio, you simply create the folder structure in your project, add your files, and run WSPBuilder as a post-build process or independently of your build process. WSPBuilder automatically creates the manifest.xml file and bundles it into a solution file to deploy the content of the \12 folder.

If your Visual Studio solution contains multiple projects containing Features and assemblies, you simply point the SolutionPath option to your Visual Studio solution folder and WSPBuilder will package all the components into a single SharePoint solution. The need to create a specific folder structure within your project may require extensive modification to existing projects and may be confusing to developers not familiar with Features or solutions.

Bear in mind that open source tools like WSPBuilder are constantly evolving and regression testing may not be as robust as that of commercial tools. Consequently, an update to the tool could impact features that worked in an earlier release. For instance, the addition of the feature to build solutions in WSPBuilder changed the behavior of the existing SolutionPath option and required changing it to ProjectPath; otherwise an empty solution would be created.

A new version of WSPBuilder, WSPBuilder Extensions, is available as an add-in to Visual Studio 2005 and Visual Studio 2008. The extensions add two WSPBuilder project types

and several item types, such as Web Part with Feature and Feature with Receiver. Adding an item creates the necessary folder structure and files needed for packaging and deployment, including the .WebPart file for Web Part items. The extensions also add deployment options to the project menu, with the limitation that you can only deploy locally.

Currently, there is limited documentation available for this release; however, the author walks through using the extensions in his blog. The project is located at codeplex.com/wspbuilder.

SPDeploy

The SPDeploy tool is an appealing blend of what SharePoint development teams are looking for in a solution packaging and deployment tool. What's really appealing about this tool is its focus on remote deployment. With it, you create your .wsp file from a standard C# class library project inside Visual Studio 2005 and it automatically reads the tree of files in Visual Studio and builds your .wsp package. The tool supplies an extension to MSBuild and incorporates .targets files to provide a reliable incremental build and deployment process.

Deployment targets are executed at the command prompt and can also be launched from Visual Studio 2005 with some minor modifications. Targets include CompileWsp, AddWsp, DeployWsp, UpgradeWsp, RetractWsp, DeleteWsp, UpgradeIncremental, UpgradeIncrementalFiles, UpgradeIncrementalAssembly, CreateSite, DeleteSite, CreateWebApplication, and DeleteWebApplication.

The author of the tool, Clint Simon, provides some useful tips on remote debugging. Also, be sure to read the blog post titled "SharePoint Custom Application Development Methodologies" at ascentium.com/blog/sp/default.aspx. It's the most accurate view we have seen for development teams of more than one.

STSDev

Another promising deployment tool emerging from the CodePlex community is STSDev (Simple Tools for SharePoint 2007 Development). The project is coordinated by SharePoint guru Ted Pattison, who wrote about it in the March 2008 issue of MSDN Magazine (msdn2.microsoft.com/magazine/cc337895.aspx). This tool provides a good

foundation for building, testing, and debugging .wsp files and is very useful for development teams who need to settle on a common process for their packaging and deployment efforts.

STSDev is a console application that initiates different dialog boxes and uses a custom .targets file to build .wsp files. While the tool is not integrated into Visual Studio, it can be shelled out to Visual Studio by pointing to the .exe file through the external tools interface. Once you make the Visual Studio connection, the tool automates the process of creating and updating the manifest.xml and .ddf files and adds common SharePoint developer commands to the Visual Studio environment.

STSDev creates the basic Visual Studio solution project structure or deployment files directory that populates the requisite SharePoint files needed to generate the .wsp file. It also creates the solutionconfig.xml file used to modify the input parameters for the manifest.xml file. The Microsoft.SharePoint.targets file provides a number of different commands that allow you to choose and switch between different deployment configurations.

This is a great tool for developers who are new to SharePoint development and the CodePlex project site contains three excellent tutorials worth viewing. You can access the project at codeplex.com/stsdev/Release/ProjectReleases.aspx?ReleaseId=10119.

Ethan Wilansky is a Microsoft Directory Services MVP and an enterprise architect. In his day job, he leads a development team currently focused on building custom SharePoint applications, and he serves as a directory services programming solutions SME.

Paul Olszewski is an Information Specialist on a .NET Capability team. He specializes in creating and deploying reusable component applications leveraging Microsoft technologies.

Rick Sneddon is a Senior Infrastructure Specialist for EDS in its Integration Engineering practice and an MCSE. He is focused on creating portal architectures, developing technology strategies, and advising customers on the operational aspects of deploying and supporting portal composite application solutions.