Concept: Deployment

This chapter describes the deployment process of customizations in SharePoint in general as well in detail. First the components involved are described. Second the process is described. Third elements like Web Parts are described in detail.

1.1 Solutions

SharePoint comes with a solution framework[1] you need to use since you need to:

- Deploy new components and upgrade old ones across different databases, folders and servers.
- Synchronize all servers in your SharePoint farm so you can keep the state consistent across all servers.

Therefore you need to **package** all your customizations in a file called solution package[2] [3]. This file is a CAB file with a .wsp extension. If you rename the extension to .cab you can browse the file.

Figure 1: Solution Package

1.1.1 Manifest

Each solution package contains a manifest.xml since the solution framework needs to know **what** to deploy and **where** to deploy.

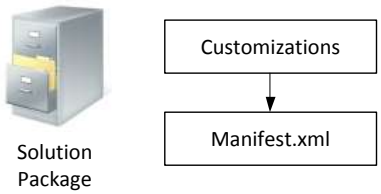

Figure 2: Solution Package with manifest.xml

[1] Deploy customizations - overview,
http://technet.microsoft.com/en-us/library/cc263205.aspx
[2] Deploy solution packages,
http://technet.microsoft.com/en-us/library/cc262995.aspx
[3] Solutions Overview,
http://msdn.microsoft.com/en-us/library/aa543214.aspx

A solution contains all your customizations like Web Parts or Controls. The following XML shows a manifest of a newly created project called “SharePoint2010”:

```
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
          SolutionId="f23abb02-a587-47be-9a8d-a5234d0caa59"
          SharePointProductVersion="14.0">
  <Assemblies>
    <Assembly Location="SharePoint2010.dll"
              DeploymentTarget="GlobalAssemblyCache" />
  </Assemblies>
</Solution>
```

The XML only contains a reference to an assembly which is part of the solution package and will be deployed to the Global Assembly Cache.

1.1.2 Visual Studio

In order to create solution packages including a manifest file you have two options:

- Either you can use the **Visual Studio 2010 SharePoint Tools** which automatically creates the solution and manifest for you or
- You can create a **custom Visual Studio solution** using makecab.exe with a custom manifest created by your own scripts.

I suggest using the Visual Studio 2010 SharePoint Tools which are also explained in detail later. With the SharePoint tools you can “group related SharePoint elements into a Feature, and then bundle multiple Features, site definitions, assemblies, and other files into a single package (.wsp file) to deploy to servers running SharePoint Server 2010.”[4]

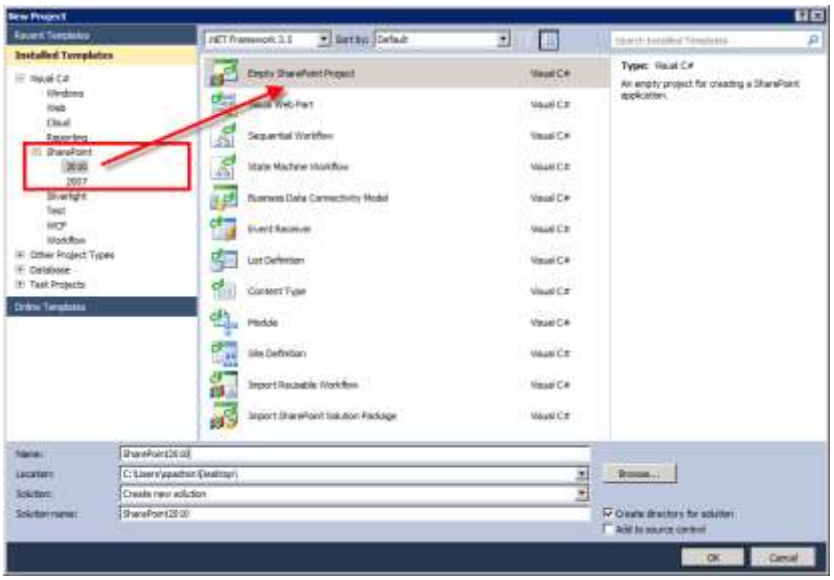

Figure 3: Visual Studio 2010 SharePoint 2010 Project

[4] Deploy solution packages, http://technet.microsoft.com/en-us/library/cc262995.aspx

1.1.3 Farm Solutions

Farm solutions allow you to interact with the whole SharePoint farm since you have full access to the server-side object model. While this gives you freedom as a developer a SharePoint administrator is limited in his actions and can't set usage limits.

Farm solutions are deployed by a farm administrator to one or more Web Applications.

Additional information

- Farm Solutions in SharePoint 2010
- Building Block: Solutions
- Differences Between Sandboxed and Farm Solutions

1.1.4 Sandboxed Solutions

Sandboxed solutions allow you to use only a subset of the server-side object model. This type of solution executes in a security restricted context that provides isolation and monitoring of the solution's code. As a developer you need to evaluate if the subset of server side code is sufficient or not. As an administrator you can have more control over a solution.

Sandboxed solutions are deployed by a Site Collection administrator to the solution gallery.

Additional information

- Sandboxed Solutions in SharePoint 2010
- Sandboxed Solutions Resource Center | SharePoint 2010

1.2 Features

A feature[5] [6] equals a customization and can be activated or deactivated in SharePoint. A feature has properties and dependencies and defines the files belonging to the customization.

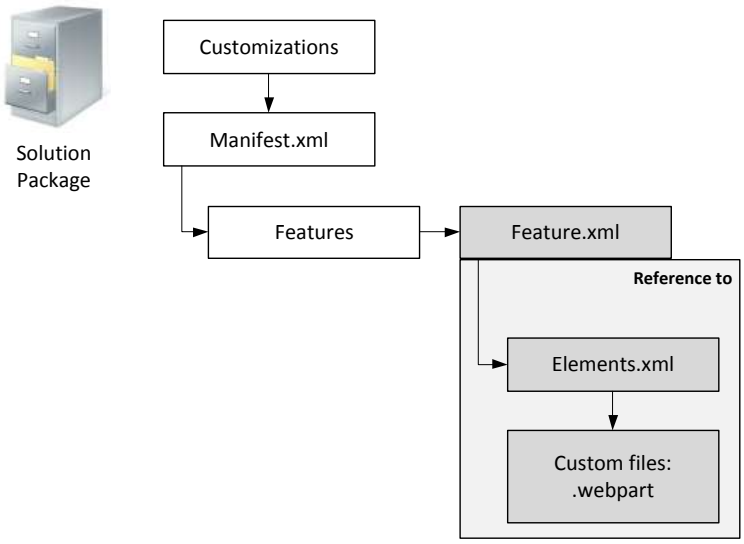

Figure 4: Feature

Example

As you can see in the picture above there is a solution containing customizations as features:

- **Feature.xml**

  The feature.xml is a definition of the feature, its properties and dependencies if needed.
- **Elements.xml**

  The feature.xml references a supporting file called elements.xml. It defines **which** custom files belong the feature and most important **what** to do with the files during deployment. Therefore the element uses types[7] to define this kind of behavior. There are examples how to use them in the chapter Practice: Development.
- **Custom files**

  In this case there is a custom Web Part definition.

[5] Deploy site elements by using Features, http://technet.microsoft.com/en-us/library/ff607680.aspx
[6] Using Features in SharePoint Foundation, http://msdn.microsoft.com/en-us/library/ms460318.aspx
[7] Element Types, http://msdn.microsoft.com/en-us/library/ms474383.aspx

The following feature XML references an element file and has a property:

```
<Feature Id="{8E16F773-B3F0-415E-9C4C-DB1DC47DE68C}"
         Title="Feature title"
         Description="Feature description"
         Scope="Site"
         xmlns="http://schemas.microsoft.com/sharepoint/">
       <ElementManifests>
             <ElementManifest Location="element.xml"/>
       </ElementManifests>
       <Properties>
             <Property Key="PropertyName" Value="PropertyValue"/>
       </Properties>
</Feature>
```

1.2.1 Scopes

A feature can have a scope meaning that its functionality is scoped or limited to the:

- **Farm**, automatically activated
- **Web Application**, automatically activated , e.g. a custom 404 error page
- **Site Collection**, e.g. Web Parts or new Master Pages and Page Layouts applying to all sites, needs manual activation
- **Web site**, e.g. a list definition used at some sub sites, needs manual activation

1.2.2 Feature Receiver

As it was written above almost every feature has an elements file and custom files. As an exception you can create features only containing a feature XML and a reference to an assembly and a class. Using this combination you can execute code while activating or deactivating the feature.

```
<Feature Id="{8E16F773-B3F0-415E-9C4C-DB1DC47DE68C}"
         Title="Feature title"
         Description="Feature description"
         Scope="Site"
         ReceiverAssembly="SharePoint2010, Version=1.0.0.0, Culture=neutral,
                           PublicKeyToken=XXXXXXXXXXXXXXXX"
         ReceiverClass="SharePoint2010.NameSpace.ClassName"
         xmlns="http://schemas.microsoft.com/sharepoint/">
       <ElementManifests />
       <Properties />
</Feature>
```

This example shows a feature scoped to a Site Collection using code.

A feature receiver is actually an event receiver which handles the following events of a feature[8]:

- Installed, Uninstalling
- Activated, Deactivating
- Upgrading

[8] Feature Events, http://msdn.microsoft.com/en-us/library/ms469501.aspx