Sitecore CMS 6.2

Presentation Component Cookbook

Tips and Techniques for CMS Administrators, Architects, and Developers

Table of Contents

Chapter 1

Introduction

This cookbook provides tips and techniques for Sitecore Administrators, Architects, and Developers working with presentation components.[1]

This document contains the following chapters:

- Chapter 1 — Introduction
- Chapter 2 — Development Infrastructure
- Chapter 3 — Layout Details
- Chapter 4 — Controls
- Chapter 5 — Layouts and Sublayouts
- Chapter 6 — RSS Features

[1] For more information about presentation components, see the Presentation Component Reference Manual at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Reference.aspx. For technical details and code samples for presentation components, see the Presentation Component XSL Reference guide at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20XSL%20Reference.aspx and the Presentation Component API Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20API%20Cookbook.aspx. For information about troubleshooting presentation components, see the Presentation Component Troubleshooting Guide at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Troubleshooting%20Guide.aspx.

Chapter 2

Development Infrastructure

This chapter provides techniques for configuring development environments to work with Sitecore layout engine presentation components.

This chapter contains the following sections:

- Requirements Analysis
- ASP.NET
- The Developer Center
- Microsoft Visual Studio

2.1 Requirements Analysis

To minimize development effort, perform thorough requirements analysis before implementation, especially to identify reusable data, application, and presentation components. Document information architecture, presentation, and functional requirements, and then map those requirements to components including data templates, workflow, security, presentation, insert options, and other properties. Document the key for each placeholder and the caching options for each presentation component. Design presentation components to support output caching by the fewest possible criteria.[2]

2.1.1 Name and Path Conventions

Each component of the project should follow standardized naming conventions. Every project should have a name, for example `MyWebSite`. Many projects also have an acronym, or a short name, for example `MWS` or `mws` for `MyWebSite`.

Developers often use the project acronym as the ASP.NET tag prefix for the project as described in the section ASP.NET Tag Prefixes. Administrators often use the project name or acronym in file system paths and the names of other objects. For example, an administrator may install the MyWebSite project to the `C:\inetpub\sitecore\MyWebSite` or `C:\inetpub\sitecore\MyWebSite\MWS` folder.

Developers may store layout and sublayout files for the MyWebSite project in the `/layout/mws` directory, with corresponding definition items within `/sitecore/layout/layouts/mws` and `/sitecore/layout/sublayouts/mws`.

Developers may store XSL rendering files for the MyWebSite project in the `/xsl/mws` directory, with corresponding definition items within `/sitecore/layout/renderings/mws`.

Developers may locate other resources, such as JavaScript and CSS files, in a directory named after the project or project acronym, or use the project name or acronym in the file names.

An administrator may name the relational databases using the project name or acronym as a prefix. For example, an administrator may name the default databases associated with the MyWebSite project mwsCore, mwsMaster, and mwsWeb.

Developers may use the project name or acronym in the name of .NET assemblies. For example, if the `MyWebSite` solution involves a single .NET assembly, a developer might configure the Visual Studio project to generate an assembly named `MyWebSite.dll`. If the `MyWebSite` solution involves multiple .NET assemblies, developers might configure the Visual Studio projects to generate several assemblies with file names that start with the `mws` prefix, such as `mwh.bl.dll` for business logic and `mwh.web.dll` for Web components.

In general, each .NET assembly should contain classes in a common namespace. For example, the `MyWebSite.dll` assembly or the `mwh.bl.dll` assembly could contain classes in various namespaces within the `MyWebSite` namespace, while the `mwh.web.dll` assembly might only contain classes in the `MyWebSite.Web.UI` namespace.

Important
To simplify administration, such as duplicating resources from one Sitecore instance to another, store presentation component definition items in project-specific folders corresponding to the path to the file. For example, for the layout file `/layouts/mws/Web.aspx`, use the layout definition item `/Sitecore/Layout/Layouts/mws/Web`.

[2] For more information about Sitecore output caching, see the Sitecore Presentation Component Reference manual at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Reference.aspx.

Important
Sitecore definition items and elements in `web.config` reference .NET components using namespace, class, and assembly names. Choose and maintain appropriate names to avoid the potential need for reconfiguration in the future.

Tip
Use namespaces that correspond to namespaces in the .NET framework or Sitecore APIs. For example, Microsoft uses the `System.Web.UI.WebControls` namespace for Web controls, and Sitecore uses `Sitecore.Web.UI.WebControls`. Consider using the equivalent of `MyWebSite.Web.UI.WebControls` for your custom Web controls.

Note
By convention, developers typically store both layouts and sublayouts within the `/layouts` file system directory.

2.2 ASP.NET

This section provides procedures and considerations for working with ASP.NET.

2.2.1 ASP.NET Tag Prefixes

ASP.NET Web forms (Sitecore layouts) and Web user controls (Sitecore sublayouts) uses tag prefixes to map tokens to namespaces in assemblies. By default, ASP.NET maps the tag prefix `asp` to the `System.Web.UI.WebControls` namespace in the `System.Web` assembly, exposing Web controls. For example:

```
<asp:textbox runat="server" />
```

When ASP.NET encounters this code, it creates an object from the `System.Web.UI.WebControls.TextBox` class.

You can use ASP.NET tag prefix registration directives in `web.config`, in each Web form, and in each Web user control to map additional tag prefixes to any namespace in any assembly. It is not necessary to map the `asp` or the `sc` tag prefix.

You can make tag prefixes available to all layouts and sublayouts using `/configuration/system.web/pages/controls/add` elements in `web.config`. For example, Sitecore uses this approach by default for the `asp` and `sc` tag prefixes:

```
…
<system.web>
  <pages validateRequest="false">
  <controls>
    <add tagPrefix="sc" namespace="Sitecore.Web.UI.WebControls"
      assembly="Sitecore.Kernel"/>
    ...
```

You can register tag prefixes in individual layouts and sublayouts. The following example code demonstrates registration of the `mws` tag prefix to enable use of classes in the `MyWebSite.Web.UI.WebControls` namespace implemented in the `MyWebSite.dll` assembly:

```
<%@ Register TagPrefix="mws" Namespace="MyWebSite.Web.UI.WebControls"
Assembly="MyWebSite" %>
```

Note
When a developer drags a Web control onto a layout or sublayout in the Developer Center, Sitecore adds the appropriate registration directive to the layout or sublayout using the tag prefix specified in the Web control definition item. When a developer drags a Web control from the Visual Studio Toolbox onto a Web form or Web user control in Visual Studio, Visual Studio adds a tag prefix registration directive, but uses an arbitrary tag prefix.

2.2.2 ASP.NET Control Identifiers (IDs)

Sitecore's layout engine uses ASP.NET. ASP.NET structures pages as hierarchies of literal controls that result in static content and server controls that generate dynamic content. Sitecore layouts, sublayouts, and renderings are ASP.NET controls. Each control must have a unique ASP.NET control identifier within the page.

2.2.3 Code-Behind, Code-Beside, or CodeFile?

ASP.NET Web forms and Web user control can separate design from logic using code-behind. Code-behind separates markup and controls in an `.aspx` or `.ascx` file from logic in a separate .NET code-behind file. For example, the Web user control `sublayout.ascx` could have the C# code-behind file `sublayout.ascx.cs`.

The `CodeBehind` attribute of the `Page` directive (for a layout) or `Control` directive (for a sublayout) references the code-behind file. For example:

```
        <%@ Control Language="C#" AutoEventWireup="true" CodeBehind="MySublayout.ascx.cs"
          Inherits="Namespace.Web.UI.MySublayout" %>
```

ASP.NET 2.0 introduces partial classes. Implementing a layout or sublayout as a partial class creates a third file called the designer file, for example `MySubLayout.ascx.designer.cs`. ASP.NET 2.0 also introduces the `CodeFile` attribute as an alternative to the `CodeBehind` attribute. For example:

```
        <%@ Control Language="c#" AutoEventWireup="true"
Inherits="Namespace.Web.UI.MySublayout"
          CodeFile="/layouts/MyWebSite/MySublayout.ascx.cs" %>
```

One potential advantage of using the `CodeFile` attribute is that the code-behind file can be compiled at runtime instead of being precompiled by a developer.

By default, Visual Studio 2008 creates Web forms and Web user controls using partial classes with the `CodeBehind` attribute in the `Page` or `Control` directive.

By default, the Developer Center creates sublasyouts without code-behind. If the developer chooses to create code-behind for a sublayout, the Developer Center does not create a partial class and uses the `CodeFile` attribute. The Developer Center does not support creating layouts with code-behind.

Tip
While developing code-behind for a layout or sublayout, you can use the `CodeFile` attribute to avoid the performance impact of compilation, which clears caches by restarting ASP.NET. Be sure to change the attribute name from `CodeFile` attribute to `CodeBehind` and compile before moving the component out of development, to avoid the need to copy the code file into the production environment.

2.3 The Developer Center

The **Developer Center** is a browser-based application for working with Sitecore presentation and other components. The **Developer Center** provides the features and functionality of an Integrated Development Environment (IDE) such as Microsoft Visual Studio, but runs in a Web browser instead of running as a Windows application.

Important
The **Developer Center** requires Microsoft Internet Explorer 6 or higher. Sitecore recommends Internet Explorer (IE) 8.[3]

Important
Both Internet Explorer and the **Developer Center** define a **File** menu. Unless otherwise specified, all references to the **File** menu in this document refer to the **File** menu in the **Developer Center**, not the **File** menu in **Internet Explorer**.

Important
While the **Developer Center** provides tools for manipulating tables, you can develop components using tables, CSS, or both.

Note
You can edit layouts, sublayouts, and XSL renderings using the **Developer Center**, or any text editor. Sitecore recommends C# using Visual Studio 2008 with the Web Application project model.

The **Developer Center** allows you to edit layouts, sublayouts, and XSL renderings using only a Web browser.

The **Developer Center** does not require client license or client installation.

The **Developer Center** is less intimidating and easier to learn than Visual Studio.

The **Developer Center** provides easy access to Sitecore's browser-based debugger.

2.3.1 How to Access the Developer Center

To access the Developer Center from within the Sitecore Desktop:

1. In Internet Explorer, access the Sitecore login page (`/sitecore`).
2. In Internet Explorer, in the Sitecore login page, in the **User Name** and **Password** fields, enter authentication credentials.
3. In Internet Explorer, in the Sitecore login page, click **Options** until the Sitecore user interface choices appear.
4. In Internet Explorer, in the Sitecore login page, double-click **Desktop**. The Sitecore desktop appears in the browser.
5. In Internet Explorer, in the Sitecore Desktop, click the **Sitecore** button, and then click **Developer Center**. The **Developer Center** appears in the Sitecore Desktop.

2.3.2 How to Access Recently-Used Items in the Developer Center

The **Developer Center Start Page** provides shortcuts to open layouts, sublayouts, XSL renderings, and other recently-used items.

[3] For more information about Sitecore Web client requirements, see the Sitecore CMS Installation Guide at http://sdn.sitecore.net/Products/Sitecore%20V5/Sitecore%20CMS%206/Installation.aspx. For instructions to configure Internet Explorer, see the Internet Explorer Configuration guide at http://sdn.sitecore.net/reference/Sitecore%206/IE%20Configuration%20Reference.aspx.

To use shortcuts in the **Developer Center** to access recently-used items:

1. In the **Developer Center**, click the **View** menu, and then click **Startpage**.
2. In the **Developer Center**, in the **Recent Files** panel, click the recently used item. The recently used item appears in the **Developer Center**.

2.3.3 How to Access the Content Editor from within the Developer Center

To access the Content Editor from within the Developer Center:

1. In the **Developer Center**, click the **View** menu, and then click **Content Explorer**. The **Content Explorer** appears in the **Developer Center**.

1. In the **Developer Center**, in the **Content Explorer**, double-click any item. **Content Editor** appears in the **Developer Center** with that item selected.

2.3.4 The Developer Center Code Boilerplate Files

When you create a layout, sublayout, or XSL rendering, or other type of code asset in the Developer Center, Sitecore creates a definition item and copies a code boilerplate file to create the new code file.[4] Sitecore stores these boilerplate files in the `/sitecore/shell/templates` folder in the document root of the Web site.

Important
Before and after updating a boilerplate file, consider adding that file to a Visual Studio project and any source code management system in use. Remember to hide all files in Visual Studio Solution Explorer before debugging.

Consider updating the boilerplate files Developer Center uses:

- To cause all new layouts or sublayouts to contain an ASP.NET tag prefix.
- To cause all new layouts or sublayouts to inherit from a common base class or other properties.
- To register custom namespaces in all new XSL renderings.
- To redefine the `$home` variable in all new XSL renderings, or to remove it.
- To include XSL template libraries in all new XSL renderings.

Disable (comment) code that may not be applicable to every use of the boilerplate file. For example, an XSL rendering may contain a commented `<xsl:include>` XSL element for a template library that may not be needed in every XSL rendering, which you can uncomment easily when needed.

How to Edit the Developer Center Boilerplate Files

You can edit the boilerplate files that the Developer Center uses to create layouts, sublayouts, and XSL rendering in Visual Studio or any text editor, or in the Developer Center. To edit the boilerplate files using the Developer Center, click the File menu, and then click Open File. The file selection dialog appears.

The boilerplate files in `/sitecore/shell/templates` contain the following:

- **`layout.aspx`**: boilerplate for layout files.

[4] For more information about the boilerplate file for new XSL renderings, see the Presentation Component XSL Reference at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20XSL%20Reference.aspx.

- **`layout.aspx.cs`**: boilerplate for layout code files.
- **`sublayout.ascx`**: boilerplate for sublayout files.
- **`sublayout.aspx.cs`**: boilerplate for sublayout code files.
- **`xsl.xslt`**: boilerplate for XSL rendering files.

2.4 Microsoft Visual Studio

This section provides information for developers working on Sitecore solutions working with Microsoft Visual Studio. The following sections assume that the reader is familiar with Microsoft Visual Studio 2005 or higher.

Important
All references to .NET namespaces and class names in this document are case-sensitive. XML, XSL, and XPath are also case-sensitive.

Note
This document describes C# and Microsoft Visual Studio 2008. User interface steps for other languages or versions of Microsoft Visual Studio may differ.

Microsoft Visual Studio provides a single development environment for all types of files including ASP.NET, XSL, CSS, JavaScript, and other resources.

Microsoft Visual Studio provides IntelliSense, syntax completion, automatic code indentation, error underlining, and other integrated development environment features.

Microsoft Visual Studio provides a debugger for .NET code.

Microsoft Visual Studio integrates with source code management tools.

Note
The ASP.NET 2.0 Web Application project model available in Visual Studio 2005 Service Pack 1 and Visual Studio 2008 is appropriate for most Sitecore solutions.[5]

Tip
When you create a layout, sublayout, or XSL rendering in the Developer Center, Sitecore invokes a wizard that copies a boilerplate file to the new location and creates a corresponding definition item. This process involves fewer steps than creating the item in Visual Studio and then manually creating the corresponding definition item in Sitecore.

2.4.1 How to Show Visual Studio Solution Explorer

To show the Microsoft Visual Studio **Solution Explorer**:

1. In Visual Studio, open the Web Application project.
2. In Visual Studio, click the **View** menu, and then click **Solution Explorer**.

2.4.2 How to Show or Hide All Files in Visual Studio Solution Explorer

To show all files in Solution explorer, or hide files that are not included in the project, in the Web Application project, show Visual Studio **Solution Explorer**, and then toggle **Show All Files** (typically the second button from the left at the top of Visual Studio **Solution Explorer**). For instructions to show Visual Studio **Solution Explorer**, see the section How to Show Visual Studio Solution Explorer.

Important
Because the Microsoft Visual Studio 2008 debugger may stop responding when showing all files, show all files, add files to the project, and then hide all files.

[5] For instructions to create a Visual Studio Web Application project for use with a Sitecore solution, see the Presentation Component Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx.

2.4.3 How to Create a Visual Studio Web Application Project

Sitecore supports the Visual Studio Web Application project model for Sitecore solutions.

Note
The initial release of Visual Studio 2005 did not include the Web Application project model. Visual Studio 2005 Service Pack 1 includes the Web Application project model.[6]

Important
Create a Visual Studio solution and at least one project for each Sitecore solution that uses Visual Studio. Follow the steps outlined below as a single sequence once for each new Sitecore solution, then add new projects to the existing solution as appropriate.

To create a Visual Studio Web Application project for an existing Sitecore solution:

1. In Visual Studio, click the **File** menu, then click **New**, and then click **Project**. The new project dialog appears.
2. In the new project dialog, in the **Project Types** tree, expand **Visual C#**, and then click **Web**.
3. In the new project dialog, in the **Templates** list, click **ASP.NET Web Application**.
4. In the new project dialog, in the **Name** field, enter the name of the project, which is typically the name of the project, for example `MyWebSite`. Visual studio will use this name as the default .NET namespace and assembly name.
5. In the new project dialog, in the **Location** field, enter the document root of the Sitecore solution, for example `C:\inetpub\sitecore\MyWebSite\WebSite`.
6. Disable **Create directory for solution**.
7. Accept the default value for **Location**, and then click **OK**. **Visual Studio** creates the project in a subdirectory of the directory specified by **Location**, and then opens that project.

To move the Visual Studio project:

1. Close Visual Studio.
2. Using the Windows file system explorer, navigate to the directory containing the project, for example `C:\inetpub\sitecore\MyWebSite\WebSite\MyWebSite`.
3. Move the `Properties` folder, the `.csproj` file, and the `.csproj.user` file to the document root of the Sitecore solution. For example, move `C:\inetpub\sitecore\MyWebSite\WebSite\MyWebSite\Properties`, `C:\inetpub\sitecore\MyWebSite\WebSite\MyWebSite\MyWebSite.csproj`, and `C:\inetpub\sitecore\MyWebSite\WebSite\MyWebSite\MyWebSite.csproj.user` to `C:\inetpub\sitecore\MyWebSite\WebSite`.

Note
If you did not disable **Create directory for solution** when you created the Visual Studio project, then move the `Properties` directory and the `.csproj` and `.user` files from the `C:\inetpub\sitecore\MyWebSite\WebSite\MyWebSite\MyWebSite` directory to the `C:\inetpub\sitecore\MyWebSite\WebSite` directory.

4. Delete the other file system resources created by Visual Studio. For example, delete the entire `C:\inetpub\sitecore\MyWebSite\WebSite\MyWebSite` folder.

To remove the moved project from the Visual Studio start page:

1. In Visual Studio, click the **View** menu, then click **Other Windows**, and then click **Start Page**. The **Recent Projects** list appears.

[6] For more information about Visual Studio 2005 Service Pack 1, see http://msdn.microsoft.com/en-us/vstudio/bb265237.aspx.

2. In Visual Studio, in the **Recent Projects** list, click the project name. Visual Studio prompts whether to remove the entry from the list of recent projects.

To open the Visual Studio Web Application Project:

1. In Visual Studio, click the **File** menu, then click **Open**, and then click **Project/Solution**. A file selection dialog appears.
2. In the file selection dialog, navigate to the directory containing the Visual Studio project, for example `C:\inetpub\sitecore\MyWebSite\WebSite`.
3. Click the `.csproj` file, and then click **Open**. For example, click `MyWebSite.csproj`, and then click **Open**.

To create a Visual Studio solution for the Web Application project, close Visual Studio. Visual Studio prompts to save changes to the solution file, for example `MyWebSite.sln`.

To configure the Visual Studio Web Application project:

1. In Visual Studio, open the Web Application project, and show Visual Studio **Solution Explorer**.
2. In Visual Studio **Solution Explorer**, right-click `default.aspx`, and then click **Exclude from Project**. This file is part of Sitecore, not the Visual Studio solution.
3. In Visual Studio **Solution Explorer**, right-click **References**, and then click **Add Reference**. An a reference selection dialog appears.
4. In the reference selection dialog, click the **Browse** tab. A file selection dialog appears.
5. In the file selection dialog, navigate to the `/bin` folder within the document root of the Sitecore solution, for example `C:\inetpub\siotecore\MyWebSite\WebSite\bin`.
6. In the file selection dialog, click `Sitecore.Kernel.dll`, and then click **OK**.
7. In Visual Studio **Solution Explorer**, expand **References**, then right-click `Sitecore.Kernel`, and then click **Properties**. The Properties pallet appears in Visual Studio.
8. In the Visual Studio **Properties** pallet, set the **Copy Local** property of the `Sitecore.Kernel` assembly reference to **False**.

Warning
If the **Copy Local** property of a reference to an assembly in the `/bin` folder in the document root of the Sitecore solution is not **False**, then Visual Studio can delete assemblies from the `/bin` folder, which can cause the Sitecore solution to fail. Set the **Copy Local** property to False for each reference to an assembly in the `/bin` folder within the document root of the Web site.

Important
Add additional assembly references only when necessary.

To configure the assembly name and default namespace for a Visual Studio Web Application project:

1. In Visual Studio, open the Web Application project.
2. In Visual Studio, click the **Project** menu, and then click the **Properties** option for the project. For example, click from the **Project** menu, and then click **MyWebSite Properties**. The project properties editor appears in Visual Studio.
3. In the project properties editor, click the **Application** tab.
4. In the project properties editor, for **Assembly name**, enter a name for the assembly for the project to generate, without the `.dll` extension. For example, enter `MyWebSite`.
5. In the project properties editor, for **Default Namespace**, enter the default namespace for the project. For example, enter `MyWebSite`.

2.4.4 How to Add an Existing File to a Web Application Project

To add an existing layout, sublayout or XSL rendering file to a Visual Studio Web Application project:

1. In Visual Studio, open the Web Application project, and show Visual Studio **Solution Explorer**.
2. In Visual Studio **Solution Explorer**, show all files.
3. In Visual Studio **Solution Explorer**, expand the directory containing the file.
4. In Visual Studio **Solution Explorer**, right-click the file, and then click **Include In Project**.

Important
In Visual Studio **Solution Explorer**, always hide all files or the Visual Studio debugger may become unresponsive.

2.4.5 How to Add Sitecore Controls to the Visual Studio Toolbox

You can add the Sitecore controls to the Visual Studio Toolbox.

Important
If you work with a version of Sitecore other than that used to add the controls to the toolbox, when you drag a Sitecore control from toolbox onto a layout or sublayout, Visual Studio may overwrite various assemblies in the `/bin` folder with the version from the installation used to add the controls to the toolbox. Whenever working with a version of Sitecore other than that used to add controls to the toolbox, remove the controls from the toolbox, and add them again.

Note
By default, Visual Studio adds all of the controls in the assembly when you add an assembly to the Visual Studio **Toolbox**. Most developers do not use most of these controls. Removing unused controls from the Visual Studio **Toolbox** makes Visual Studio faster and makes it easier for developers to locate controls

To add Sitecore controls to the Visual Studio toolbox:

1. In Visual Studio, close any open solutions or projects.
2. In Visual Studio, click the **Tools** menu, and then click **Choose Toolbox Items**. The control selection dialog appears.
3. In the control selection dialog, click **Browse**. The file selection dialog appears.
4. In the file selection dialog, navigate to the `/bin` folder within the document root of the Web site, and then click the `Sitecore.Kernel.dll` assembly.
5. In the file selection dialog, click **Open**. The control selection dialog selects and highlights all of the controls in the assembly.
6. In the control selection dialog, with all of the controls in the assembly selected and highlighted, clear the checkbox next to any one of the controls to clear the checkboxes for all controls in that assembly.
7. In the control selection dialog, sort by **Namespace**.
8. In the control selection dialog, scroll to the `Sitecore.Web.UI.WebControls` namespace.
9. In the control selection dialog, select the checkboxes next to each of the Sitecore controls to add to the toolbox, most commonly `FieldRenderer` (the FieldRenderer Web control), `Method` (the method rendering Web control), `Sublayout`, `WebPage` (the URL rendering Web control), and `XslFile` (the XSL rendering Web control).

To group the Sitecore controls within a single tab in the Visual Studio toolbox:

1. In Visual Studio, show the Visual Studio **Toolbox**.

2. In Visual Studio, right-click in the Visual Studio **Toolbox**, and then click **Add Tab**. For the tab name, enter **Sitecore**.
3. In the Visual Studio **Toolbox**, drag the Sitecore controls from the **General** tab onto the **Sitecore** tab.

To remove the Sitecore controls from the Visual Studio toolbox:

1. In Visual Studio, show the Visual Studio **Toolbox**.
2. In Visual Studio, right-click the Visual Studio **Toolbox**, and then click **Choose** Items. The control selection dialog appears.
3. In the control selection dialog, sort by **Namespace**.
4. In the control selection dialog, scroll to the `Sitecore.Web.UI.WebControls` namespace.
5. In the control selection dialog, clear the checkboxes next to each of the Sitecore controls.

-or-

1. In Visual Studio, close any open projects and solutions, and then show the Visual Studio **Toolbox**.
2. In Visual Studio, right-click the Visual Studio **Toolbox**, and then click **Reset Toolbox**.

2.4.6 How to Debug .NET Code Using Visual Studio

Use the Sitecore log files, administrative pages, and the Sitecore browser-based debugger to identify components containing logical errors, performance bottlenecks, and other unfavorable code conditions. Then use the Visual Studio debugger to debug the .NET code.

To debug .NET code using Visual Studio:

1. To ensure the ASP.NET worker process is active, use a Web client to request an ASP.NET resource from the Sitecore solution, such as the home page.
2. In Visual Studio, show Visual Studio **Solution Explorer**, and hide all files.
3. In Visual Studio, navigate to the appropriate line in the relevant .NET code file.
4. To create a breakpoint in Visual Studio, click the **Debug** menu, and then click **Toggle Breakpoint**, or press F9.
5. In Visual Studio, click the **Debug** menu, and then click **Attach to Process** (or press CTRL-ALT-P). The **Attach to Process** dialog appears.
6. In the **Attach to Process** dialog, select **Show processes from all users** and **Show processes in all sessions**.
7. In the **Attach to Process** dialog, click the `aspnet_wp.exe` process on IIS 5 (Windows XP) or the `w3wp.exe` process on IIS 6 or 7 (Windows 2003, Windows Vista, or Windows 2008), and then click **Attach**.
8. Use a Web client such as Microsoft Internet Explorer to request an ASP.NET resource that uses the code to debug.

Top stop debugging:

- In Microsoft Visual Studio, click the **Debug** menu, and then click **Stop Debugging**.

Important
Do not click **Start Debugging** after clicking the **Debug** menu in Visual Studio. This would start the Casini Web server installed by Microsoft Visual Studio rather than using IIS.

2.4.7 How to Create a Collection of Web Service Methods

All requests for resources that access Sitecore run in a Sitecore context that includes a logical site definition (`/configuration/sitecore/sites/site` in `web.config`) providing access to resources such as configuration. The location of the Web service file (`.asmx`) on the file system determines the context site and hence the context in which the Web service request executes.

The URL for the default Sitecore Web service library is`/sitecore/shell/webservice/service.asmx`. These Web service methods run in the context of the site `/configuration/sitecore/sites/site element` in `web.config` with `name shell`. This request context sets `Sitecore.Context.Database` to the Core database and `Sitecore.Context.ContentDatabase` to the Master database. Most of the default Web service methods accept a parameter indicating the name of the database the service should access instead of relying solely on configuration.

Requests for ASP.NET resources within the `/sitecore modules/web` directory run in the context of the logical site named `modules_website`, which by default sets `Sitecore.Context.Database` to the Web database and leaves `Sitecore.Context.ContentDatabase` null. Place the Web service file in a directory that sets the context databases as required for the Web service, or pass the database name to the Web service as a parameter and avoid using `Sitecore.Context.Database`.

To add a collection of Web service methods to an ASP.NET Web Application project in Visual Studio:

1. In the Visual Studio Web Application project, in Visual Studio **Solution Explorer**, create a folder, add a folder to the project, or locate an existing folder already in the project.
2. In Visual Studio **Solution Explorer**, right click the folder identified in the previous step, then click **Add**, and then click **New Item**. The **Add New Item** dialog appears.
3. In the **Add New Item** dialog, in the **Templates** list, click **Web Service**.
4. In the **Add New Item** dialog, for **Name**, enter a name for the file that will contain the Web service methods, and then click **Add**. Visual Studio creates the Web service file.
5. In Microsoft Visual Studio, in the Web service file, create Web service methods.

2.4.8 How to Optimize Visual Studio Performance

This section provides tips to improve the performance of Visual Studio.

Before closing Visual Studio, close user interface components that do not need to be open the next time you start the application.

To access the Visual Studio **Options** dialog:

1. In Visual Studio, click the **Tools** menu, and then click **Options**.

To disable Visual Studio RSS feeds:

1. In the Visual Studio **Options** dialog, expand **Environment**, and then click **Startup**.
2. In the Visual Studio Options dialog, clear **Start Page news channel**.
3. In the Visual Studio Options dialog, disable **Download content every**.

To disable the start page:

1. In the Visual Studio **Options** dialog, expand **Environment**, and then click **Startup**.
2. In the Visual Studio Options dialog, for **At Startup drop-down**, select **Show empty environment**.

Note
To show the Visual Studio start page, in Visual Studio, click the **View** menu, then click **Other Windows**, and then click **Start Page**.

To disable the Visual Studio splash screen:

1. In the Windows Desktop, right-click the shortcut you use to start Visual Studio, and then click **Properties**. A properties dialog appears for the shortcut.
2. In the shortcut properties dialog, click the **Shortcut** tab.
3. In the shortcut properties dialog, on the Shortcut tab, add the `/nosplash` command line option to **Target**. For example:

```
"C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\devenv.exe" /nosplash.
```

To disable animation:

1. In the Visual Studio **Options** dialog, click **Environment**.
2. In the Visual Studio **Options** dialog, clear the **Animate environment tools** checkbox.

To disable change tracking:

1. In the Visual Studio **Options** dialog, click **Text Editor**.
2. In the Visual Studio **Options** dialog, disable **Track changes**.

To turn off active item tracking:

1. In the Visual Studio **Options** dialog, click **Projects and Solutions**.
2. In the Visual Studio **Options** dialog, disable **Track Active Item in Solution Explorer**.

To turn off AutoToolboxPopulate:

1. In the Visual Studio **Options** dialog, click **Windows Forms Designer**.
2. In the Visual Studio **Options** dialog, under **Toolbox**, set **AutoToolboxPopulate** to **False**.

To cause Visual Studio to open Layouts and Subalyouts in source code view rather than design view by default:

1. In the Visual Studio **Options** dialog, click **HTML Designer**.
2. In the Visual Studio Options dialog, for **Start pages in**, select **Source View**.

If you use Jetbrains ReSharper, then pressing CTRL-F12 provides update the list of methods and fields above the editing pane.[7] To disable the Visual Studio navigation bar that provides equivalent functionality, or if you do not use the Visual Studio navigation bar:

1. In the Visual Studio **Options** dialog, expand **Text Editor**, and then click **C#.**
2. In the Visual Studio **Options** dialog, disable the **Navigation bar**.

[7] For more information about JetBrains ReSharper, see http://www.jetbrains.com/resharper.

Chapter 3

Layout Details

This chapter provides procedures for working with layout details.

This chapter contains the following sections:

- How to Work with Layout Details
- How to Reset Layout Details to Standard Values
- How to Copy Layout Details
- How to Determine Presentation Components Used
- Working with Devices

3.1 How to Work with Layout Details

This section provides procedures for working with layout details.

Note
You can also use the Design Pane of Page Editor to edit layout details.

Important
Sitecore stores layout details in a field defined in the standard template from which all other data templates inherit. Like all field values, layout details defined in individual items override layout details defined in the standard values of the data template associated with the item. To reduce data duplication and administration, assign layout details in standard values instead of individual items. If different items based on a common data structure require different layout details, consider a new data template that inherits from the existing data template, and using its standard values to define layout details.

Important
For backwards compatibility with previous versions, Sitecore applies layout details defined in the data template definition item if there are no layout details in the item or the standard values item associated with its data template. Define layout details in the standard values item associated with the data template definition item, not in the data template definition item itself.

Important
Always define layout details for the default device, which Sitecore activates by default for all incoming HTTP requests that do not specify an alternate device.

Note
The term control includes sublayouts and all types of renderings.

3.1.1 The Device Editor

The **Device Editor** allows you to select a device when entering layout details.

How to Access the Device Editor

To access the **Device Editor**:

1. In the **Template Manager** or the **Content Editor**, edit the standard values item or the individual item.
2. In the **Template Manager** or the **Content Editor**, click the **Presentation** tab.
3. In the **Template Manager** or the **Content Editor**, on the **Presentation** tab, in the **Layout** group, click the **Details** command. The **Layout Details** dialog appears.
4. In the **Layout Details** dialog, below the device for which you want to configure layout details, click **Edit**. The **Device Editor** appears.

How to Select a Layout

To select a layout to use for a device:

1. In the **Device Editor**, click the **Layout** tab. For instructions to access the **Device Editor**, see the section How to Access the Device Editor.
2. In the **Device Editor**, with the **Layout** tab selected, for **Layout** down, select the layout.

How to Add a Control

To add a control in layout details:

1. In the **Device Editor**, click the **Controls** tab, and then click **Add**. The **Select a Rendering** dialog appears. For instructions to access the **Device Editor**, see the section How to Access the Device Editor.
2. In the **Select a Rendering** dialog, select the rendering.
3. To open rendering properties after adding the rendering to layout details, in the **Select a Rendering dialog**, select **Open Properties after this dialog closes**. For more information about rendering properties, see the section Controls.
4. In the **Select a Rendering** dialog, click **Select**.

How to Order Controls

To order controls in layout details:

1. In the **Device Editor**, click the **Controls** tab, and then click a control to select it. For instructions to access the **Device Editor**, see the section How to Access the Device Editor.
2. In the **Device Editor**, on the **Controls** tab, with the control selected, click **Move Up** or **Move Down** to order the control relative to other controls.

Important
The order of presentation components in layout details controls the order in which the layout engine creates and binds them to the control hierarchy. Sort sublayouts that contain placeholders before the controls that bind to those placeholders.

Note
When multiple presentation components bind to a single placeholder, their order in layout details controls the order of the markup written to the output stream.

How to Remove a Control

To remove a control from layout details:

1. In the **Device Editor**, click the **Controls** tab, and then click a control to select it. For instructions to access the **Device Editor**, see the section How to Access the Device Editor.
2. In the **Device Editor**, on the **Controls** tab, with the control selected, click **Remove**. Sitecore removes the rendering from the layout details.

How to Replace a Control

To replace the rendering associated with existing rendering properties, without changing any of those properties:

1. In the **Device Editor**, click the **Controls** tab, then click the control to select it. For instructions to access the **Device Editor**, see the section How to Access the Device Editor.
2. In the **Device Editor**, on the **Controls** tab, with the control selected, click **Change**. The **Select a Rendering** dialog appears.
3. In the **Select a Rendering** dialog, click a rendering to select it, and then click **Select**. Sitecore replaces the rendering in the layout details.

3.2 How to Reset Layout Details to Standard Values

To reset the layout details for an item to those defined in standard values item of the data template associated with the item:

1. In the **Content Editor**, select the item for which to reset layout details to those defined in the standard values item of the data template associated with the item.
2. In the **Content Editor**, click the **Presentation** tab.
3. In the **Content Editor**, on the **Presentation** tab, in the **Layout** group, click the **Reset** command. Sitecore prompts for confirmation before resetting the layout details field in the selected item to its standard value.

3.3 How to Copy Layout Details

To copy layout details from one item to another:

1. In the **Content Editor**, select to the source item from which to copy layout details.
2. In the **Content Editor**, click the **Presentation** tab.
3. In the **Content Editor**, on the **Presentation** tab, in the **Details** group, click the **Layout** command. The **Layout Details** dialog appears.
4. In the layout details dialog, click the **Copy To** link under any device. The **Copy Device** dialog appears.
5. In the **Copy Device** dialog, for **Target Devices**, select the devices for which to copy layout details.
6. In the **Copy Device** dialog, for **Target Item**, click the item to which you will copy layout details, and then click **Copy**. Sitecore copies the layout details for the selected devices from the source item to the target item.

To copy layout details from a single source item to multiple target items:

1. In the **Content Editor**, select to the source item.
2. In the **Content Editor**, show standard fields and raw values.[8]
3. In the **Content Editor**, with the source item selected, in the **Layout** section, triple-click the value of the **Renderings** field to select that value, and then copy that value to the Windows clipboard.
4. In the **Content Editor**, for each target item, in the **Layout** section, triple-click the value of the **Renderings** field to select that value, and then overwrite that value with the contents of the Windows clipboard.
5. In the **Content Editor**, hide raw field values and the standard template fields.

[8] For instructions to show or hide the standard template fields and raw values, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Client%20Configuration%20Cookbook.aspx.

3.4 How to Determine Presentation Components Used

You can use the Sitecore debugger to determine the presentation components used to service an HTTP request.

Important
The Sitecore debugger accesses the Web database by default. Publish changes to layout details before debugging.

To determine the presentation components used to render a page:

1. In the Sitecore debugger, show the ribbon.
2. In the Sitecore debugger, in the **Rendering** group, select **Information**.
3. In the Sitecore debugger, in the **Trace** group, click the Activate command.
4. In the Sitecore debugger, hover over the information icons (the green triangles) and investigate details about the various controls used in the page.
5. In the Sitecore debugger, scroll to the **Sitecore Trace** section and look for errors resulting from invalid layout details, such as no rendering definition item corresponding to an ID reference or no placeholder matching a specified placeholder key.

3.5 Working with Devices

This section provides procedures for working with Sitecore devices.

3.5.1 How to Create a Device

To create a device:

1. In the **Content Editor**, select the `/Sitecore/Layout/Devices` item.
2. In the **Content Editor**, with the `/Sitecore/Layout/Devices` item selected, insert a device definition item using the `/System/Layout/Device` data template.
3. If the new device should have a fallback device, then in the **Content Editor**, in the device definition item, in the **Data** section, in the **Fallback device** field, select the fallback device.
4. Define device activation criteria as described in the section, How to Define Device Activation Criteria.

Important
In the device item, in the Data section, do not select the Default checkbox.

3.5.2 How to Define Device Activation Criteria

After you create a device, define criteria that cause Sitecore to activate that device for incoming HTTP requests. Sitecore can activate a device for HTTP requests that contain a specific query string parameter, or that match a specific Web client user agent string. You can configure each managed site to use a different device. You can implement an `httpRequestBegin` pipeline processor to activate a device.

To configure a query string parameter to activate the device, in the **Content Editor**, in the device definition item, in the **Detection** section, for **Query string**, enter the query string `key=value` pair. For example, to activate the device when a URL contains the query string parameter `x` with a value of `1`, enter `x=1`.

To configure a specific Web client user agent to activate the device, in the **Content Editor**, in the device definition item, in the **Detection** section, for **Browser** agent, enter the user agent to match. For example, enter `blackberry` to activate the device for all blackberry clients. Comparison is not case-sensitive.

To configure the layout engine to activate the device for all HTTP requests associated with a specific managed Web site, set the `device` attribute in the `/configuration/sitecore/sites/site` element in `web.config` that defines that site to the name of that device.[9]

To use other criteria to activate the device, implement an `httpRequestBegin` pipeline processor to replace or follow `Sitecore.Pipelines.HttpRequest.DeviceResolver`, and set `Sitecore.Context.Device`.[10]

[9] For more information about configuring multiple logical sites, see http://sdn.sitecore.net/Articles/Administration/Configuring%20Multiple%20Sites.aspx.
[10] For more information about using .NET APIs to activate a device, see the Presentation Component API Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20API%20Cookbook.aspx.

Chapter 4

Controls

This chapter provides procedures for working with controls. In the context of this document, the term controls includes placeholders, sublayouts, XSL renderings, Web controls, URL renderings, method renderings, and the FieldRenderer Web control.

This chapter contains the following sections:

- How to View the Output of a Control
- Rendering Definition Items
- How to Access Control Properties
- How to Set Control Properties Using Visual Studio
- Common Control Properties
- Placeholders
- Sublayouts
- The FieldRenderer Web Control
- XSL Renderings
- Web Controls
- Method Renderings
- URL Renderings

4.1 How to View the Output of a Control

To view the output of a single control using the Sitecore debugger:

1. In the Sitecore debugger, show the ribbon.
2. In the Sitecore debugger, in the **Rendering** group, select **Information**.
3. In the Sitecore debugger, hover over the information icon (the green triangle) representing the rendering. A rendering information panel appears.
4. In the rendering information panel, click the **Output** tab. The output of the rendering appears in the rendering information panel.

4.2 Rendering Definition Items

You must insert a rendering definition item before you can use a rendering in Sitecore user interfaces including the Developer Center, the Device Editor, and the design pane of the Page Editor. For instructions to insert the various types of rendering definition items, see the sections Sublayouts, XSL Renderings, Web Controls, Method Renderings, and URL Renderings.

Important
You can use the **Description** field in the **Data** section of each rendering definition item to provide information about the rendering to users. You can enter a description of the output of the component and some example output. For components that support caching, include a reminder to set caching options. For URL renderings that do not specify a URL and method renderings that do not specify a method, enter a reminder for the user to enter these properties. Otherwise, enter the URL or the name of the method.

Important
Before inserting a rendering definition item, use the **Content Editor** and the Windows file system Explorer to create Sitecore folders and file system directories to contain the definition item and the file that the definition item will reference.

Tip
If you work with a presentation component only in Visual Studio and not in the Sitecore user interfaces, then it is not necessary to register that component in Sitecore.

4.3 How to Access Control Properties

To access the rendering properties dialog for a control bound statically to a layout or sublayout using the **Developer Center**:

1. In the **Developer Center**, open the layout or sublayout.
2. In the **Developer Center**, in the layout or sublayout, click the **Design** tab.
3. In the **Developer Center**, in the layout or sublayout, double-click the control. The rendering properties dialog appears.

To access the **Control Properties** dialog for a control bound dynamically to a placeholder using layout details:

1. In the **Device Editor**, click the **Controls** tab, then click the control to select it. For instructions to access the **Device Editor**, see the section How to Access the Device Editor.
2. In the **Device Editor**, click **Edit**. The **Control Properties** dialog appears.

4.4 How to Set Control Properties Using Visual Studio

To set control properties for a control bound statically to a layout or sublayout using Visual Studio:

1. In the Visual Studio Web Application project, open the layout or sublayout.
2. In Visual Studio, in the layout or sublayout, right-click the control, and then click **Properties**. The Visual Studio Properties pallet appears.
3. Use the Visual Studio Properties pallet to set properties, or click the **Source** tab and enter attributes and values for the control definition element.

4.5 Common Control Properties

This section describes properties common to various types of controls.

Important
For each control that you bind statically to a layout or sublayout, define the `id` attribute as an ASP.NET control identifier. For more information about ASP.NET control identifiers, see the section ASP.NET Control Identifiers (IDs).

4.5.1 How to Configure Control Caching Options

This section provides instructions to configure caching for each control.

Important
Configure caching options whenever you use a control, whether you bind the control dynamically to a layout or sublayout, or dynamically to a placeholder.

Important
Caching options in the rendering definition item do not apply to renderings bound dynamically using layout details. Define caching options for each control in layout details.

Important
When you adds a rendering to a layout or sublayout using the **Developer Center**, Sitecore copies caching options from the rendering definition item to properties of the control in the layout or sublayout. Define caching options in each rendering definition item.

Important
When you update a rendering definition item, Sitecore does not update layouts and sublayouts to which you have statically bound that rendering. After updating caching options in a rendering definition item, remember to update caching options for that control in the layouts and sublayouts that statically bind that rendering.

Important
To avoid unnecessary memory consumption, avoid caching the same output twice. For example, if you cache the output of a sublayout, then you may not need to cache the output of each rendering used by that sublayout.

To configure caching options using the **Control Properties** dialog:

- In the **Control Properties** dialog, in the **Caching** section, select caching options.

To configure caching options using the rendering properties dialog:

- In the rendering properties dialog, click the **Caching** tab, and then select caching options.

To configure caching options using Visual Studio:

1. In Visual Studio, in the layout or sublayout, click the **Design** tab, then right-click the control, and then click **Properties**. The Visual Studio Properties pallet appears.
2. In the Visual Studio **Properties** pallet, configure caching options.

4.5.2 How to Configure the Data Source of a Control

To pass a data source item using the **Control Properties** dialog:

- In the **General** section, in the **Data Source** field, enter the full path to the data source item.

-or-

- Click the **Insert Link** command above the field, and then select an item.[11]

To pass a data source item to a control using Visual Studio:

1. In the Visual Studio Web Application project, in the layout or sublayout, click the **Design** tab, then right-click the control, and then click **Properties**. The Visual Studio **Properties** pallet appears.
2. In the Visual Studio **Properties** pallet, for the **DataSource** property, enter the full path to an item.

4.5.3 How to Pass Rendering Parameters to a Control

To pass parameters to a control using the **Control Properties** dialog:

- In the **Control Properties** dialog, enter control properties.

To pass parameters to a control using the rendering properties dialog:

- In the rendering properties dialog, click the **Parameters** tab, and then enter named parameters.[12]

Important
To set properties of a Web control in the rendering properties dialog, including the `FieldName` property of the FieldRenderer Web control, use the **Parameters** tab, not the **Attributes** tab. You can use the parameter named `Parameters` to set the `Parameters` property of the Web control to a URL-encoded sequence of key=value pairs.

To pass parameters to a control using Visual Studio:

1. In the Visual Studio Web Application project, in the layout or sublayout, click the **Design** tab, then right-click the control, and then click **Properties**. The Visual Studio **Properties** pallet appears.
2. In the Visual Studio **Properties** pallet, for the **Parameters** property, enter named parameters using URL-escaped `key=value` pairs, separated by ampersand characters ("`&`").

[11] To access the data source passed to a sublayout, see http://trac.sitecore.net/SublayoutParameterHelper.
[12] To access the parameters passed to a sublayout, see http://trac.sitecore.net/SublayoutParameterHelper.

4.6 Placeholders

The only property specific to placeholders is the placeholder key.

Note
Sitecore placeholders are instances of a Web control. The **Developer Center** uses the same dialogs to set the properties of all Web controls, including Placeholders. For placeholders, the **Placeholder** field in the rendering properties dialog is irrelevant. In the rendering properties dialog, enter the key of the placeholder as on the **Parameters** tab as a parameter named `key`. In layout details, for each control, in the **General** section, in the **Placeholder** field, enter the placeholder key or fully qualified placeholder key to which the control should bind.

Important
When assigning a placeholder key in a layout or sublayout, always use an individual placeholder key such as `content`, not a fully qualified placeholder key such as `/main/content`.

4.7 Sublayouts

For the `path` attribute of the `<sc:sublayout>` control, enter the path to the Web user control file relative to the document root of the Web site. [13]

[13] For information about APIs that you can use in presentation components including Web controls, see the Presentation Component API Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20API%20Cookbook.aspx.

4.8 The FieldRenderer Web Control

For the `FieldName` property of a FieldRenderer Web control, enter the name of the field for the control to process.

Note
The FieldRenderer Web Control retrieves the value of the specified field from the context item by default. Pass a data source to the FieldRenderer Web control to retrieve the value from a specific item.

Note
The FieldRenderer Web control does not support output caching options. To support output caching for this control, create a Web control that inherits from `Sitecore.Web.UI.WebControls.FieldRenderer`. Override the `GetCachingID()` method to return a cache key for the control, for example a string containing the GUID of the item and the GUID of the field definition item, plus any additional information to identify the caching context, and use this Web control instead of the default FieldRenderer Web control.

4.9 XSL Renderings

This section provides guidance for working with XSL renderings in both the **Developer Center** and in Visual Studio.[14]

In XSL rendering definition items, in the Data section, the Path field contains the path to the XSL rendering file relative to the document root of the Web site.

Important
Avoid inline .NET code in XSL renderings in favor of custom .NET XSL extensions.

Note
The system invokes XSL rendering transformations on the server, not on Web clients.

4.9.1 How to Create an XSL Rendering

To create an XSL rendering:

1. In the **Developer Center**, click the **File** menu, and then click **New**. The **New File** dialog appears.
2. In the **New File** dialog, in the **Categories** tree, click **Renderings**.
3. In the **New File** dialog, in the **Templates** list, click **XSLT File**, and then click **Create**.
4. In the **New File** dialog, for **Name**, enter a name for the XSLT rendering, and then click **Next**. Sitecore use this name for both the XSLT rendering definition item and the XSLT rendering file.
5. In the **New File** dialog, in the content tree, click the item that will contain the XSL rendering definition item, and then click **Next**.
6. In the **New File** dialog, in the file system tree, click the directory that will contain the XSL rendering file, and then click **Create**. The XSL rendering appears in the **Developer Center**.

4.9.2 How to View the Output of an XSL Rendering

Developers can analyze the output of an individual XSL rendering using the Sitecore debugger, or using the previewing pane in the **Developer Center**.

To view the output of an XSL rendering in the **Developer Center**:

1. In the **Developer Center**, open an existing XSL rendering.
2. In the **Developer Center**, at the top of the editing window, enable **Preview**.
3. In the Developer Center, in the previewing pane below the editing pane, in the drop-down list, select an item. The **Developer Center** will pass this item to the XSL transformation engine as the data source of the rendering (`$sc_item` and `$sc_currentitem`).
4. In the **Developer Center**, click the **Refresh** button at the top right of the previewing window. The previewing window shows the result of invoking the XSL transformation with the selected item as the data source.

4.9.3 The XSL Rendering Boilerplate File

When you create a new XSL rendering using the Developer Center, Sitecore duplicates the XSL rendering boilerplate file, which provides a basis for the new code.[15]

[14] For more information about XSL renderings, see the Presentation Component XSL Reference at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20XSL%20Reference.aspx.

The Main XSL Template Block

After you create an XSL rendering, add your code to the main XSL template.

```
<xsl:template match="*" mode="main">
  <!--//TODO:enter XSL rendering code here-->
</xsl:template>
```

Note
In XSL, blocks of code contained within `<xsl:template>` XSL elements are called XSL templates.

4.9.4 Custom XSL Template Libraries

XSL templates are blocks of XSL code enclosed in the `<xsl:template>` XSL element that function similar to methods, functions, and procedures in other programming languages.[16]

Developers use XSL templates to contain reusable blocks of XSL code. Developers can use XSL templates procedurally by invoking them by name, or declaratively by invoking them through XPath match patterns.

The context element for each XSL template is the element that was the context element at the point that the XSL transformation engine invoked the XSL template.

XSL templates can accept a variable number of named parameters using the `<xsl:param>` and `<xsl:with-param>` XSL elements.

How to Create an XSL Template Library

To create an XSL template library:

1. Using Windows File System Explorer or Visual Studio, create or navigate to the file system directory that will contain the XSL template library code file, for example `/xsl/mywebsite`.
2. Create a new `.xslt` file, for example `library.xslt`. Use the following prototype:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sc="http://www.sitecore.net/sc"
  xmlns:dot="http://www.sitecore.net/dot"
  exclude-result-prefixes="dot sc">
  <xsl:template name="TemplateName">
    <!--//TODO:logic-->
  </xsl:template>
</xsl:stylesheet>
```

3. Use `<xsl:param>` and `<xsl:with-param>` to pass parameter to XSL template blocks:

```
<xsl:template name="TemplateName" />
  <xsl:param name="ParamName" select="'DefaultValue'" />
  <!--logic->
</xsl:template
...
<xsl:call-template name="TemplateName">
  <xsl:with-param name="ParamName" select="'ParamValue'" />
</xsl:call-template>
```

4. Use variables to contain the results of an XSL template block:

---

[15] For more information about the boilerplate file used for new XSL renderings, see the Presentation Component XSL Reference at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20XSL%20Reference.aspx.

[16] For more information about XSL template libraries, see the Presentation Component XSL Reference at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20XSL%20Reference.aspx.

```
<xsl:template>
  <xsl:choose>
    <xsl:when test="//TODO:logic">
      Variable Value
    </xsl:when>
    <xsl:otherwise>
      Default Value
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:variable name="VariableName">
  <xsl:call-template name="TemplateName" />
</xsl:variable>
```

How to Reference an XSL Template Library in an XSL Rendering

To reference an XSL template library in an XSL rendering:

1. Edit the XSL rendering.
2. Add a line of code such as the following below the existing `<xsl:output>` element:

```
<xsl:include href="/xsl/mywebsite/library.xslt" />
```

3. Replace `library.xslt` with the URL of the XSL template library file.

Note
The `href` attribute of the `<xsl:include>` element is a reference to a URL. This URL can be the full path to the XSL template library file from the document root of the IIS Web site, a relative path such as `library.xslt` or `../library.xslt`, or even a fully qualified URL including a hostname.

To enable developers to easily add this reference to any rendering they create, see the section The XSL Rendering Boilerplate File. Add a commented reference to the library in the boilerplate file used for new XSL rendering using the appropriate value for the `href` attribute:

```
<!--<xsl:include href="library.xslt" />-->
```

Note
Sitecore 6 introduces support for XSL extension controls such as `<sc:text>` in XSL template libraries. There is no need to limit code in XSL template libraries to XSL extension methods such as `sc:fld()`.

Important
In cases where performance is critical or you require the same logic in both XSL and .NET renderings, .NET XSL extension libraries are more appropriate than XSL template libraries.

4.9.5 Custom XSL Extension Methods

XSL extensions methods written in .NET provide an optional solution in cases where XSL would be cumbersome, perform poorly, present other disadvantages, or simply cannot support a requirement.[17] Consider XSL extension libraries:

- To access data stored in an external source such as a database or application other than a Sitecore database.
- To access .NET APIs.
- To perform resource-intensive operations.
- To increase readability of the code used for complex operations

[17] For more information about XSL extension methods, see the Presentation Component XSL Reference at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20XSL%20Reference.aspx.

Note
For a more complete explanation the advantages and disadvantages of XSL relative to other rendering technologies, see the Presentation Components Reference manual.

Tip
Before implementing an XSL extension in .NET, investigate the default XSL extensions provided by Sitecore to determine if the required functionality already exists.

Note
In addition to XSL extension methods, Sitecore supports XSL extension controls. This document does not describe XSL extension controls. XSL extension methods are more flexible than XSL extension controls. XSL extension controls use the angle-bracket syntax, such as the following:

```
<sc:text field="FieldName">.
```

The equivalent using only XSL extension methods:

```
<xsl:value-of select="sc:field('FieldName',.)" disable-output-escaping="yes" />
```

How to Create an XSL Extension Method Library Class

To create an XSL extension library class, in the Visual Studio Web Application project, create a class containing methods to represent each of the custom XSL extension methods. This class must have a constructor that accepts no parameters. XSL extension methods typically return strings or objects of type `System.Xml.XPath.XPathNodeIterator`, which generally represent items in the database, data retrieved from an external system, or data generated dynamically.

How to Register a Custom XSL Extension Method Library

You can register any class as a custom XSL extension method library. The class does not need to implement any interface or inherit from any specific base class.

To register a .NET class as an XSL extension library:

1. In `web.config`, navigate to the `/configuration/sitecore/xslExtensions` element.
2. Within the `<xslExtensions>` element, insert a new line based on the following:

```
<extension mode="on" type="Namespace.Class, Assembly"
namespace=http://domain.tld/class
  singleInstance="true"/>
```

3. Replace the values of the `type` and `namespace` attributes with the appropriate class signature and URL.

Note
The value of the `namespace` attribute must be a valid, unique URL, but it does not have to be a valid Web page.

To use the new extension in XSL renderings and the boilerplate file used for XSL renderings, map the namespace to the URL and the `exclude-result-prefixes` attribute of the rendering.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sc="http://www.sitecore.net/sc"
  xmlns:dot="http://www.sitecore.net/dot"
  xmlns:namespace="http://www.domain.tld/class"
  exclude-result-prefixes="dot sc namespace">
```

Tip
Consider adding the namespace definition to the boilerplate file used for XSL renderings. For more information about boilerplate files, see the section The Developer Center Code Boilerplate Files.

How to Use a .NET XSL Extension Library

To use a .NET XSL extension library:

1. Edit the XSL rendering file.
2. In the `<xsl:stylesheet>`, element add an attribute mapping a namespace to the URL associated with the extension. Add the namespace to the exclude-result-prefixes attribute, which contains a list of namespaces separated by whitespaces. For example:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:sc="http://www.sitecore.net/sc"
xmlns:dot="http://www.sitecore.net/dot"
xmlns:my="http://mydomain.tld/myclass"
exclude-result-prefixes="dot sc my">
```

Important
If you do not add the namespace to the value of the `exclude-result-prefixes` attribute, the XSL transformation engine may output attributes that are not valid by the HTML specification.

3. Use this namespace to invoke methods in the XSL extension library. For example, if the class contains the method `MyMethod()` that accepts a `string` parameter and returns a `string`, that method can be used to populate an XSL variable:

```
<xsl:variable name="myvariable" select="my:MyMethod('MyParameterValue')" />
```

Alternatively, the rendering can write that string directly to the output stream:

```
<xsl:value-of select="my:MyMethod('MyParameterValue')" />
```

Tip
Consider adding the custom namespace definition to the boilerplate file used for XSL renderings as described in the section The XSL Rendering Boilerplate File.

To use the new extension in XSL renderings and the boilerplate file used for XSL renderings, map the namespace to the URL and the `exclude-result-prefixes` attribute of the rendering.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sc="http://www.sitecore.net/sc"
  xmlns:dot="http://www.sitecore.net/dot"
  xmlns:namespace="http://www.domain.tld/class"
  exclude-result-prefixes="dot sc namespace">
```

Important
Add the namespace to the `exclude-result-prefixes` attribute. Otherwise, the generated markup may contain the namespace.

Tip
Consider adding the namespace definition to the boilerplate file used for XSL renderings.

How to Add Methods to the sc Namespace

To add methods to the `sc` namespace, override `Sitecore.Xml.Xsl.XslHelper`:

1. Create a class that inherits from `Sitecore.Xml.Xsl.XslHelper`.
2. In `web.config`, in the `/configuration/sitecore/xslExtensions/extension` element with `namespace` `http://www.sitecore.net/sc`, replace the value of the `type` attribute with the signature of your class. The `sc` namespace then exposes the methods in your class as well as the methods in the `Sitecore.Xml.Xsl.XslHelper` base class.

```
<extension mode="on" type="Namespace.Class,Assembly"
  namespace="http://www.sitecore.net/sc" singleInstance="true" />
```

How to Access Properties of an XSL Extension Class

To access properties of an XSL extension class library object, use explicit `get_Property()` and `set_Property()` methods. For example:

```
<xsl:if test="get_PropertyName()">
  <xsl:value-of select="set_PropertyName('PropertyValue')" />
</xsl:if>
```

In this case, the `<xsl:value-of>` XSL element sets the property, and does not generate any output.

4.9.6 XSL Extension Method Examples

This section contains examples of custom .NET XSL extension methods.

GetHome(): Return a Sitecore.Data.Items.Item

The boilerplate file for XSL renderings defines a variable named `$home` using an XPath statement. This variable is invalid if you use an XSL rendering on a site that does not have `/Sitecore/Content/Home` as its start item. You can use an XSL extension method to determine the home item using logic rather than hard-coding a path.

First determine the home item for the site. Then use the `Sitecore.Configuration.Factory.GetItemNavigator()` method to convert the `Sitecore.Data.Items.Item` to the `System.Xml.XPath.XPathNodeIterator` representation used by XSL renderings.

```
namespace Namespace.Xml.Xsl
{
  private Sitecore.Data.Items.Item GetHomeItem()
  {
    Sitecore.Data.Database db = Sitecore.Context.Database;
    Sitecore.Data.Items.Item home = database.GetItem(Sitecore.Context.Site.StartPath);
    return(home);
  }
  public class XslHelper
  {
    public Sitecore.Xml.XPath.ItemNavigator GetHome()
    {
      return(Sitecore.Configuration.Factory.CreateItemNavigator(GetHomeItem()));
    }
    public string GetHomeID()
    {
      return(GetHomeItem().ID.ToString());
    }
  }
}
```

Update the `$home` variable definition in XSL rendering and the boilerplate file used for new XSL renderings.

```
<xsl:variable name="home" select="namespace:GetHome()" />
```

Note
It is generally more efficient to process a string than it is to process an XML structure. When possible, use a method that returns an ID as a `string` instead of returning an item as a `System.Xml.XPath.XPathNavigator`. For example, unless you are already using the `$home` variable and just need to update the logic used to define that variable, avoid defining the `$home` variable. When possible, use the `GetHomeID()` method instead of `GetHome()`. If you update the XSL rendering boilerplate file as suggested above, comment out this variable declaration to avoid unnecessary overhead. Developers can uncomment this line if they need this variable.

GetRandomSiblings(): Return Multuple Values Using XML

You can return a list from an XSL extension using a delimited string, or using XML. You can use this technique to return a list of item IDs, which you can process using XSL code similar to that used with the `sc:Split()` XSL extension method.

For example, a rendering needs to generate links to five random siblings of the context item, but never to the context item itself, and without ever generating two links to the same sibling. The following extension library class inherits from the `Sitecore.Xml.Xsl.XslHelper` class in order to use its `GetItem()` method to retrieve the `Sitecore.Data.Items.Item` corresponding to a `System.Xml.XPath.XPathNodeIterator`.

```
namespace Namespace.Xml.Xsl
{
  public class XslHelper : Sitecore.Xml.Xsl.XslHelper
  {
    public XPathNodeIterator GetRandomSiblings(XPathNodeIterator iterator,int max)
    {
      Sitecore.Xml.Packet packet = new Sitecore.Xml.Packet("values","");
      iterator.MoveNext();
      Sitecore.Data.Items.Item item = GetItem(iterator);
      if(item != null )
      {
        Sitecore.Collections.ChildList children = item.Parent.Children;
        if(children.Count>1)
        {
          if(max>children.Count-1)
          {
            max = children.Count-1;
          }
          List<Sitecore.Data.ID> ids = new List<Sitecore.Data.ID>();
          Random rand = new Random();
          while(ids.Count<max)
          {
            int index = rand.Next(children.Count);
            if(children[index].ID!=item.ID && !ids.Contains(children[index].ID))
            {
              packet.AddElement("value",children[index].ID.ToString());
              ids.Add(children[index].ID);
            }
          }
        }
      }
      XPathNavigator navigator = packet.XmlDocument.CreateNavigator();
      if (navigator == null)
      {
        navigator = new XmlDocument().CreateNavigator();
      }
      navigator.MoveToRoot();
      navigator.MoveToFirstChild();
      return navigator.SelectChildren(XPathNodeType.Element);
    }
  }
}
```

This code will return an XML structure such as the following:

```
<values>
  <value>{ID}</value>
  ...
  <value>{ID}</value>
</values>
```

You can process this structure using code such as the following:

```
<xsl:for-each select="namespace:GetRandomSiblings(.,5)">
  <xsl:for-each select="sc:item(text(),$sc_currentitem)">
    <sc:link>
      <xsl:value-of select="@name" />
      <br />
    </sc:link>
  </xsl:for-each>
</xsl:for-each>
```

Note
This code is provided only for demonstration purposes and could be very inefficient with a small number of siblings.

4.10 Web Controls

This section provides procedures for working with Web controls.[18]

For the name of a Web control definition item, use the name of the Web control class.

Web controls depend on several mandatory parameters.

For the **Tag** property of a Web control, enter name of the Web control class, such as `MyWebControl`.

For the **Tag Prefix** property of a Web control, enter the ASP.NET tag prefix for the namespace containing the Web control, such as `mws`.

For the **Namespace** property of a Web control, enter the namespace containing the Web control class, such as `MyWebSite.Web.UI.WebControls`.

For the **Assembly** property, enter the name of the assembly containing the Web control class, without the `.dll` extension, such as `MyWebSite`.

4.10.1 How to Create a Web Control Class

To create a Web control class:

1. In the Visual Studio Web Application project, create a class using the following prototype:

```
Namespace.Web.UI.WebControls
{
  public class ClassName : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(HtmlTextWriter output)
    {
      //TODO: write to output
    }
    protected override string GetCachingID()
    {
      return GetType().ToString();
    }
  }
}
```

2. In the class, replace `Namespace.Web.UI.WebControls` with the appropriate namespace to contain the class.
3. In the class, replace `ClassName` with the name of the class.
4. In the class, replace `//TODO: write to output` with logic to write to the `output HtmlTextWriter`.
5. In the class, replace `GetType().ToString()` with logic to return a cache key for the control.

4.10.2 How to Register a Web Control

You can register a Web control using a wizard in the **Developer Center**, or using the **Content Editor**.

Tip
Use the wizard to register the first Web control in a namespace. To register additional Web controls in the same namespace, use the **Content Editor** to duplicate an existing Web control definition item, and then update values in the new item.

[18] For information about APIs that you can use in presentation components including Web controls, see the Presentation Component API Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20API%20Cookbook.aspx.

To register a Web control using the **Developer Center**:

1. In the **Developer Center**, click the **File** menu, and then click **New**. The **New File** dialog appears.
2. In the **New File** dialog, in the **Categories** tree, expand **Renderings**.
3. In the **New File** dialog, in the **Templates** list, click **Web Control**, and then click **Create**. The **Create WebControl Wizard** appears.
4. In the **Create WebControl Wizard**, enter Web control properties.
5. In the **Create WebControl Wizard**, click **Test**, resolve any issues until the **Developer Center** successfully finds the Web control. Then click **Next**.
6. In the **Create WebControl Wizard**, click the folder that will contain the Web control definition item, and then click **Create**. The Web control definition item appears in the **Developer Center**.

To register a Web control using the **Content Editor**:

1. In the **Content Editor**, navigate to the project-specific folder beneath the `/Sitecore/Layout/Renderings` item that will contain the Web control definition item.
2. In the **Content Editor**, insert a Web control definition item using the `/System/Layout/Renderings/Webcontrol` data template.
3. In the **Content Editor**, in the Web control definition item, enter the properties of the Web control.

4.10.3 How to Add a Property to a Web Control

To add a property to a Web control using Visual Studio:

1. In the Visual Studio Web Application project, open the Web control class.
2. In the class, create the property. For example, to create a `string` property named `PropertyName`:

```
public string PropertyName
{
  get { throw new NotImplementedException() };
  set { throw new NotImplementedException() };
}
```

Note
For instructions to set Web control properties, see the section How to Pass Rendering Parameters to a Control.

4.11 Method Renderings

This section provides procedures for working with method renderings.

A method rendering definition item may or may not specify a method. If a method rendering definition item does not specify a method, the user who binds the method rendering to a placeholder in layout details or drags it onto a layout or a sublayout in the **Developer Center** must specify the method. To provide default methods for different method renderings, insert multiple method renderings definition items that specify different methods.

For the name of a method rendering definition item, if the definition item will specify a method, use a name of the method, possibly including the namespace and class name. Otherwise, create the method rendering definition item under the `/Sitecore/Layout/Renderings/System` item, and use the name **Method Rendering**.

For **Method**, enter the name of the method.

For **Class**, enter the name of the class containing the method.

For **Assembly**, enter the name of the assembly containing the class, without the `.dll` extension.

Important
Consider using a Web control to wrap a method instead of using a method rendering.

Note
The method rendering Web control does not support output caching. To support output caching for method renderings, create a Web control that inherits from `Sitecore.Web.UI.WebControls.Method`, and override the `GetCachingID()` method to return a cache key for the control. For example, you could return a string containing the assembly, namespace, class, and method names, plus any additional criteria to identify the caching context, and use this Web control instead of the default method rendering Web control. Alternatively, create a Web control that invokes the method, and define the `GetCachingID()` method in that Web controls.

4.11.1 How to Create a Method Rendering Class and Method

To create a method rendering class and method:

1. In the Visual Studio Web Application project, if a class containing method renderings already exists, consider adding a method to that class. Otherwise, create a new class.
2. In the class, create a method with the following signature:

```
public string MethodName()
```

3. In the class, complete the body of the method.

4.11.2 How to Register a Method Rendering

You can register a method rendering using the **Developer Center** or the **Content Editor**.

To register a method rendering using the **Developer Center**:

1. In the **Developer Center**, click the **File** menu, and then click **New**. The **New File** wizard appears.
2. In the **New File** wizard, in the **Categories** tree, click **Renderings**.
3. In the **New File** wizard, in the **Templates** list, click **Method Rendering**, and then click **Create**. The Create Method Rendering wizard appears.
4. In the **Create Method Rendering** wizard, enter method rendering properties.
5. In the **Create Method Rendering** wizard, click **Test**, and then resolve any issues until the **Developer Center** successfully finds the method. Then click **Next**.

6. In the Create Method Rendering wizard, click the item that will contain the method rendering definition item, and then click **Create**.

To register a method rendering using the **Content Editor**:

1. In the **Content Editor**, select the item within `/Sitecore/Layout/Renderings` that will contain the method rendering definition item.
2. In the **Content Editor**, insert a method rendering definition item using the `/System/Layout/Renderings/Method Rendering` data template.
3. In the **Content Editor**, in the method rendering definition item, enter method rendering properties.

4.12 URL Renderings

This section provides procedures for working with URL renderings.

A URL rendering definition item may or may not reference a URL. If a URL rendering definition item does not specify a URL, the user who binds the URL rendering to a placeholder in layout details or drags it onto a layout or sublayout in the Developer Center must specify a URL. To provide default URLs for different URL renderings, insert multiple URL renderings definition items that specify different URLs.

For the name of a URL rendering definition item, if the definition item will specify a URL, use a name that identifies the URL. Otherwise, create the URL rendering definition item in the `/Sitecore/Layout/Renderings/System` folder, and use the name URL Rendering.

For the URL property, enter the URL to process.

Note
The layout engine invokes URL renderings on the Web server. Web clients do not invoke URL renderings directly.

4.12.1 How to Register a URL Rendering

To register a URL rendering using the **Developer Center**:

1. In the **Developer Center**, click the **File** menu, and then click **New**. The **New File** wizard appears.
2. In the **New File** wizard, in the **Categories** tree, click **Renderings**.
3. In the **New File Wizard**, in the **Templates** list, click **Url Rendering**, and then click **Create**. The **Create URL Rendering** wizard appears.
4. In the **Create URL Rendering** wizard, enter URL rendering properties.
5. In the **Create URL Rendering** wizard, click **Test**, and then resolve any issues with the URL. Then click **Next**.
6. In the **Create URL Rendering** wizard, click the item that will contain the URL rendering definition item, and then click **Create**.

To register a URL rendering using the **Content Editor**:

1. In the **Content Editor**, select the descendant of the `/Sitecore/Layout/Renderings` item that will contain the rendering definition item.
2. In the **Content Editor**, insert a URL rendering definition item using the `/System/Layout/Renderings/Url Rendering` data template.
3. In the **Content Editor**, in the URL definition item, enter URL rendering properties.

4.13 How to Implement a Rendering Settings Data Template

To implement a rendering settings data template:

1. In the **Template Manager** or the **Content Editor**, select the descendant of the `/Sitecore/Templates` item that will contain the rendering settings data template definition.
2. In the **Template Manager** or the **Content Editor**, insert a rendering settings data template definition item. Consider using the name of the rendering in the name of the rendering settings data template. Select the `System/Layout/Rendering Parameters/Standard Rendering Parameters` data template as the base template.
3. In the **Template Manager** or the **Content Editor**, in the rendering settings data template definition item, configure fields with names corresponding to properties of the .NET control or parameters to the XSL rendering.

Important
Property and parameter names cannot contain spaces and other special characters.

4. In the **Content Editor**, select the rendering definition item for which the properties apply.
5. In the **Content Editor**, in the rendering definition item, in the **Editor Options** section, in the **Parameters Template** field, select the rendering settings data template definition item.
6. In the **Content Editor**, in layout details for an item that uses the rendering, enter values for the properties or parameters.

Chapter 5

Layouts and Sublayouts

This chapter provides procedures for working with layouts and sublayouts.

This chapter contains the following sections:

- Create a Layout
- Create a Sublayout
- Add a Control to a Layout or Sublayout
- Add Code-Beside to a Layout or Sublayout
- How to Add a Layout or Sublayout Partial Class File and Replace CodeFile with CodeBehind

5.1 Create a Layout

You can create a layout using the **Developer Center** or using Visual Studio or any text editor.

5.1.1 How to Create a Layout Using the Developer Center

To create a layout using the **Developer Center**:

1. In the **Developer Center**, click the **File** menu, and then click **New**. The **New File** wizard appears.
2. In the **New File** wizard, in the **Categories** tree, click **Layouts**.
3. In the **New File** wizard, in the **Templates** list, click **Layout**, and then click **Create**. The **Create Layout** wizard appears.
4. In the **Create Layout** wizard, for **Name**, enter the name of the layout, and then click **Next**. The wizard will use this name for both the layout definition item and the layout file.
5. In the **Create Layout wizard**, click the folder that will contain the layout definition item, and then click **Next**.
6. In the **Create Layout** wizard, click the directory that will contain the layout file, and then click **Create**. The layout appears in the **Developer Center**.

5.1.2 How to Register a Web Form as a Layout

To register a Web form as a layout:

1. In the Visual Studio Web Application project, create a Web form in a subdirectory of the `/layouts` directory in the document root of the Web site, for example `/layouts/MyWebSite/MyWebLayout.aspx`.
2. In the **Content Editor**, navigate to the descendant of the `/Sitecore/Layout/Layouts` item that corresponds to the location of the Web form file, for example `/Sitecore/Layout/Layouts/MyWebSite`.
3. In the **Content Editor**, insert a layout definition item using the `/System/Layout/Layout` data template. For the name of the layout definition item, enter the name of the Web form file, without the `.aspx` extension, for example `MyWebLayout`.
4. In the **Content Editor**, in the layout definition item, ignore any exception on the **Grid Designer** tab. Click the **Content** tab.
5. In the **Content Editor**, on the **Content** tab, in the **Data** section, in the **Path** field, enter the path and name of the layout file relative to the document root of the Web site, including the `.aspx` extension, for example `/layouts/MyWebSite/MyWebLayout.aspx`.

Important
You cannot register an existing Web form as a Sitecore layout using the **Layout** command template that appears in insert options for the `/Sitecore/Layout/Layouts` item. Use the **Layout** command template to insert a layout definition item and create the corresponding layout file.

5.2 Create a Sublayout

You can use the **Developer Center** to create a sublayout, or to register a Web user control created in Visual Studio as a sublayout.

5.2.1 How to Create a Sublayout in the Developer Center

To create a sublayout in the **Developer Center**:

1. In the **Developer Center**, click the **File** menu, and then click **New**. The **New File** wizard appears.
2. In the **New File** wizard, in the **Categories** tree, click **Renderings**.
3. In the **New File** wizard, in the **Templates** list, click **Sublayout**, and then click **Create**. The **Create Sublayout** wizard appears.
4. In the **Create Sublayout** wizard, for Name, enter a name for the sublayout, and then click **Next**. The **Developer Center** uses this name for both the sublayout definition item and the sublayout file.
5. In the **Create Sublayout** wizard, click the item that will contain the sublayout definition item, and then click **Next**.
6. In the **Create Sublayout** wizard, click the directory that will contain the sublayout file.
7. In the **Create Sublayout** wizard, select **Create Associated C# Code Files** to create a code file for the sublayout.
8. Click **Create**. The sublayout appears in the **Developer Center**.

5.2.2 How to Register a Web User Control as a Sublayout

To register a Web user control as a sublayout:

1. In the Visual Studio Web Application project, create a Web user control in a subdirectory of the `/layouts` directory in the document root of the Web site, for example `/layouts/MyWebSite/MySublayout.aspx`.
2. In the **Content Editor**, navigate to the descendant of the `/Sitecore/Layout/Sublayouts` item that corresponds to the project-specific subdirectory containing the sublayout, for example `/Sitecore/Layout/Sublayouts/MyWebSite`.
3. In the **Content Editor**, insert a sublayout definition item using the `/System/Layout/Sublayout` data template. For the name of the sublayout definition item item, enter the name of the Web user control file without the `.ascx` extension, for example `MySublayout`.
4. In the **Content Editor**, in the sublayout definition item, ignore any exception on the **Grid Designer** tab. Click the Content tab.
5. In the **Content Editor**, in the sublayout definition item, on the **Content** tab, in the **Data** section, for the **Path** field, enter the path and name of the layout file relative to the document root of the Web site, including the `.ascx` extension, for example `/layouts/MyWebSite/Sublayouts/MySublayout.aspx`.

Important
You cannot register an existing Web form as a Sitecore sublayout using the **Sublayout** command template that appears in insert options for the `/Sitecore/Layout/Sublayouts` item. Use the **Sublayout** command template to insert a sublayout definition item and create the corresponding sublayout file.

5.3 Add a Control to a Layout or Sublayout

This section provides procedures to add controls to layouts and sublayouts. You can use these procedures to add a placeholder or to statically bind a sublayout or rendering to a layout or sublayout.

Important
Never allow a sublayout to bind statically or dynamically to another instance of itself, or to a control that is a descendant of that sublayout. Such binding could result in infinite recursion.

5.3.1 How to Add a Control to a Layout or Sublayout Using the Developer Center

To add a control to a layout or sublayout using the **Developer Center**:

1. In the **Developer Center**, open the layout or sublayout.
2. In the **Developer Center**, in the layout or sublayout, click the **Design** tab.
3. In the **Developer Center**, click the **View** menu, and then click **Toolbox**.
4. In the **Developer Center**, drag the control from the **Toolbox** onto the layout or sublayout.
5. In the **Developer Center**, in the layout or sublayout, double-click the control. The rendering properties dialog appears.
6. In the rendering properties dialog, apply control properties.

5.3.2 How to Add a Control to a Layout or Sublayout Using Visual Studio

To add a control to a layout or sublayout using Visual Studio:

1. In the Visual Studio Web Application project, click the **View** menu, and then click **Toolbox**.
2. In Visual Studio, open the layout or sublayout that will contain the control.
3. In Visual Studio, in the layout or sublayout, click the **Design** tab.
4. In Visual Studio, drag the control from the toolbox onto the layout or sublayout, or insert control markup. For a placeholder, use the following markup as a prototype:

```
<sc:placeholder key="" id="" runat="server" />
```

For a sublayout, use the following markup as a prototype:

```
<sc:sublayout path="/layouts/path/to/file.ascx" id="" runat="server" />
```

For an XSL rendering, use the following markup as a prototype:

```
<sc:xslfile path="/xsl/path/to/file.xslt" id="" runat="server" />
```

For a FieldRenderer Web control, use the following markup as a prototype:

```
<sc:fieldrenderer fieldname="FieldName" id="" runat="server" />
```

For a Web control, register the tag prefix as described elsewhere in this document, and use the following markup as a prototype:

```
<tagprefix:classname id="" runat="server" />
```

For a method rendering, use the following markup as a prototype:

```
<sc:method methodname="MethodName" assemblyname="AssemblyName"
  classname="Namespace.Class" id="" runat="server" />
```

For a URL rendering, use the following markup as a prototype:

```
<sc:webpage id="" runat="server" url="" />
```

5. In Visual Studio, in the layout or sublayout, right-click the control, and then click **Properties**. The Visual Studio **Properties** pallet appears.
6. In the Visual Studio **Properties** pallet, apply control properties.

Note
To support Sitecore output caching for statically bound sublayouts, use the Sitecore sublayout Web control (`<sc:sublayout>`) rather than of invoking the Web user control using ASP.NET.

Important
You cannot use named control properties to pass parameters to a sublayout bound statically. Instead, use the `Parameters` property of the Sitecore sublayout Web control (<sc:sublayout>). For more information about the Parameters property, see the section How to Pass Rendering Parameters to a Control.

5.4 Add Code-Beside to a Layout or Sublayout

When you create a layout or sublayout in the **Developer Center**, Sitecore does not create a code-beside file by default. You can add a code-beside file to an existing layout or sublayout in two ways. You can copy the text of the existing layout or sublaout file, then delete the original file, then recreate that file in Visual Studio, and then paste that text into the new file. Alternatively, you can create the code-beside and designer files manually.

5.4.1 How to Add Code-Beside to a Layout or Sublayout by Deleting the Existing File

To add a code-beside file to an existing layout or sublayout by deleting the existing file:

1. In the Microsoft Visual Studio Web Application project, open the layout or sublayout. Note the exact name and location of the layout or sublayout file.
2. In Microsoft Visual Studio, in the layout or sublayout, click the **Source** tab.
3. In Microsoft Visual Studio, in the layout or sublayout, on the **Source** tab, select all of the code in the layout or sublayout file, excluding the `Page` directive at the top of a layout file or the `Control` directive at the top of a sublayout file.
4. In Microsoft Visual Studio, in the layout or sublayout, copy the selected text to the Windows clipboard.
5. In Microsoft Visual Studio **Solution Explorer**, delete the layout or sublayout.
6. In Microsoft Visual Studio **Solution Explorer**, add a new Web Form (layout) or Web User Control (sublayout) using the original layout or sublayout file name and directory.
7. In Microsoft Visual Studio, in the layout or sublayout file, click the **Source** tab.
8. In Microsoft Visual Studio, in the layout or sublayout, select all of the text except for the `Page` or `Control` directive at the top of the file.
9. Paste the contents of the Windows clipboard to replace the selected text.

5.4.2 How to Add a Code-Beside file to a Layout or Sublayout by Creating Files

To add a code-beside file to an existing layout or sublayout by creating code-beside and designer files:

1. In the Microsoft Visual Studio Web Application project, note the name of the layout or sublayout file.

Tip
To copy the name of the layout or sublayout file to the Windows clipboard, click the file in Visual Studio **Solution Explorer**, then press F2 to rename the file, then press CTRL-C to copy the file name to the Windows clipboard, and then press ESC to cancel the rename operation.

2. In Microsoft Visual Studio **Solution Explorer**, right-click the folder containing the layout or sublayout, then click **Add**, and then click **Class**. The **Add New Item** dialog appears.
3. In the **Add New Item** dialog, for **Name**, enter the name of the layout or sublayout file including the `.cs` extension, for example `MyLayout.aspx.cs` or `MySublayout.ascx.cs`, and then click **Add**. Visual Studio creates the new class as a child of the existing layout or sublayout.
4. In Microsoft Visual Studio, in the class, enter an appropriate namespace and class name.
5. In Microsoft Visual Studio, open the layout or sublayout, and then click the **Source** tab.

6. In Microsoft Visual Studio, in the layout or sublayout, replace the existing `Page` (layout) or `Control` (sublayout) directive. For a layout, use code such as the following:

```
<%@ Page language="C#" autoeventwireup="true" inherits="Namespace.Class"
  codebehind="MyLayout.aspx.cs" %>
```

   For a sublayout, use code such as the following:

```
<%@ control language="C#" autoeventwireup="true" inherits="Namespace.Class"
 codebehind="MySubalyout.aspx.cs" %>
```

7. In Microsoft Visual Studio, in the layout or sublayout, or the `inherits` attribute, enter the namespace and class name. For the `codebehind` attribute, enter the name of the code-behind file.
8. See the following section How to Add a Layout or Sublayout Partial Class File and Replace CodeFile with CodeBehind.

5.5 How to Add a Layout or Sublayout Partial Class File and Replace CodeFile with CodeBehind

To add the layout or sublayout partial class file and replace `CodeFile` with `CodeBehind`:

1. In Microsoft Visual Studio, open the Web Application project.
2. In Microsoft Visual Studio **Solution Explorer**, right-click the `.aspx` or `.ascx` file, and then click **Include In Project**.
3. In Microsoft Visual Studio **Solution Explorer**, Right-click the `.aspx` or `.ascx` file, and then click **Convert to Web Application**. Visual Studio creates the partial class file and adds it to the project and replaces the `CodeFile` attribute in the `.aspx` or `.ascx` file with a `CodeBehind` attribute.

Chapter 6

RSS Features

This chapter provides an overview of Sitecore Really Simple Syndication (RSS) features for managed Web sites.[19]

This chapter begins with an overview of Sitecore RSS features for managed Web sites, and then describes RSS configuration options.

This chapter contains the following sections:

- Sitecore RSS Overview
- RSS Configuration

[19] For more information about RSS, see http://en.wikipedia.org/wiki/RSS_(file_format). For more information about using Sitecore RSS features, see the Content Author's Cookbook at http://sdn.sitecore.net/End%20User/Sitecore%206%20Cookbooks.aspx. For information about Sitecore client CMS feeds, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Client%20Configuration%20Cookbook.aspx.

6.1 Sitecore RSS Overview

You can syndicate Sitecore items using the Really Simple Syndication (RSS) XML format. You can define any number of RSS feeds, and each feed can include any number of syndicated items. Each feed definition item specifies syndicated items to include in the feed. For more information about syndicated items, see the section Syndicated Items.

6.1.1 Feed Definition Items

Each RSS feed consists of a feed definition item based on the `System/Feeds/RSS Feed` data template.

Tip
If the data template for an item inherits from the `System/Feeds/RSS Feed` data template, then that item can provide an RSS feed.

Layout details for the default device for each feed definition item specify the `System/Feed Delivery Layout` layout. The base class for the `System/Feed Delivery Layout` layout formats the syndicated items specified by the feed definition item as an RSS feed.

How to Create an RSS Feed

To create an RSS feed:

1. In the **Content Editor**, insert a feed definition item using the `System/Feeds/RSS Feed` data template.
2. In the **Content Editor**, in the feed definition item, in the **Data** section, in the **Items** field, select an item, or enter a Sitecore query using the `query:` prefix. The **Items** field works like the source property in a data template field definition.[20] If you select an item, the RSS feed includes the children of that item that are syndicated items. If you enter a query, the RSS feed includes the syndicated items that match that query.
3. In the **Content Editor**, in the feed definition item, in the **Data** section, in the **Title** field, enter the title of the RSS feed.
4. In the **Content Editor**, in the feed definition item, in the **Data** section, in the **Description** field, enter a description of the RSS feed.

Note
You can associate an RSS feed to an alternate URL by entering a value in the **Link** field in the **Data** section of the feed definition item. If you do not specify a value for the **Link** field, then Sitecore uses the default URL of the feed definition item as the URL of the feed. If the user clicks the title of the RSS feed, then the RSS client loads the URL specified by the field named **Link**, or the default URL of the RSS feed if the field named **Link** is empty.

5. Optionally, in the **Content Editor**, in the feed definition item, in the **Additional Metadata** section, enter additional metadata about the feed.
6. If Sitecore should cache the feed, then in the **Content Editor**, in the feed definition item, in the **Caching** section, select the **Cacheable** checkbox. In the **Cache Duration** field, enter a value to control eviction for the cache entry.

Warning
Do not cache RSS feeds that depend on authentication or authorization.

[20] For more information about the source property of data template fields and Sitecore query, see the Data Definition Reference manual at http://sdn.sitecore.net/Reference/Sitecore%206/Data%20Definition%20Reference.aspx.

Note
In the **Cache Duration** field, you can enter any value that .NET can parse using the `System.TimeSpan.Parse()` method.[21]

6.1.2 Syndicated Items

Each feed can include any number of syndicated items. Each syndicated item defines layout details for the `Feed` device. The `System/Feed Delivery Layout` layout uses layout details for the `Feed` device to format each syndicated item.

Note
Sitecore does not use the `Feed` device to format individual content items. By default, no HTTP request activates the `Feed` device. Sitecore uses the `Feed` device to configure presentation for syndicated items. The `System/Feed Delivery Layout` uses these layout details to format each syndicated item.

How to Make an Item Available for Syndication

To make an item available for syndication through RSS feeds:

1. In the **Template Manager** or the **Content Editor**, select the standard values item for the type of item to syndicate, or the individual item to syndicate.
2. In the **Template Manager** or the **Content Editor**, click the **Presentation** tab.
3. In the **Template Manager** or the **Content Editor**, on the **Presentation tab**, in the **Feeds** group, click the **Design** command. The **Set Feed Presentation** dialog appears.
4. In the **Set Feed Presentation** dialog, for **Title Field**, select the field of the data template associated with the item that contains the title to appear for that type of item in feeds.
5. In the **Set Feed Presentation** dialog, for **Body Field**, select the field of the data template associated with the item that contains the body text to appear for that type of item in feeds.
6. In the **Set Feed Presentation** dialog, for **Date Field**, select the field of the data template associated with the item that contains the date to appear for that type of item in feeds.

Important
Always manage syndication entry options using standard values. Do not use layout details to update syndication entry options for an individual item. For more information about layout details, see Chapter 3, Layout Details.

Note
The **Set Feed Presentation** dialog automatically updates syndication entry options in the standard values of the data template associated with the selected item.

You can use rendering parameters to set additional syndication entry properties. To set additional syndication properties, edit layout details for the `Feed` device, and then set rendering parameters for the FeedRenderer Web control. For more information about rendering parameters, see section How to Pass Rendering Parameters to a Control.

The `Author` parameter specifies the name of a field in the data template that contains the name of the author of the syndication entry. The `Enclosure` parameter specifies the name of a field in the data template that contains a link to a media library item to associate with the syndication entry.

[21] For more information about the `System.TimeSpan.Parse()` method, see http://msdn.microsoft.com/en-us/library/se73z7b9(VS.71).aspx.

6.2 RSS Configuration

This section describes RSS configuration settings.

6.2.1 The Feeds.MaximumItemsInFeed Setting

You can set the `value` attribute of the `/configuration/sitecore/settings/setting` element in `web.config` with `name Feeds.MaximumItemsInFeed` to control the maximum number of feed items that can appear in an RSS feed on a managed Web site.

6.2.2 The Feeds.ItemExpiration Setting

You can set the `value` attribute of the `/configuration/sitecore/settings/setting` element in `web.config` with name `Feeds.ItemExpiration` to control the duration that syndicated items appear in feeds. If the `Feeds.ItemExpiration` setting is zero ("`0`"), then syndicated items do not expire. If this value of the `Feeds.ItemExpiration` setting is a positive integer, then RSS feeds exclude syndication entries for which the value of the **Date Field** plus this number of days exceeds the current system date. For more information about the **Date Field**, see the section How to Make an Item Available for Syndication.