
Sitecore CMS 6.1

# 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

[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.

**Tip**
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.

**Important**
The Developer Center requires Microsoft Internet Explorer 6 or higher. Sitecore recommends Internet Explorer (IE) 7 or higher.[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.

### 2.3.1 How to Access the Developer Center

Sitecore developers can access the Developer Center as a standalone browser-based application, or from within the browser-based Sitecore Desktop.

To access the Developer Center as a standalone browser-based application:

1. In Internet Explorer, access the Sitecore login page (/sitecore).
1. In Internet Explorer, in the Sitecore login page, click the Advanced tab.
2. In Internet Explorer, in the Sitecore login page, click Developer Center.
3. In Internet Explorer, in the Sitecore login page, in the User Name and Password fields, enter authentication credentials, and then click Login. The Developer Center appears.

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, click the Advanced tab.
3. In Internet Explorer, in the Sitecore login page, click Desktop.
4. In Internet Explorer, in the Sitecore login page, in the User Name and Password fields, enter authentication credentials, and then click Login.

[3] For more information about Sitecore user agent (browser client) requirements and configuration instructions, see the Sitecore CMS Installation Guide at http://sdn.sitecore.net/Products/Sitecore%20V5/Sitecore%20CMS%206/Installation.aspx and the Internet Explorer Configuration guide at http://sdn.sitecore.net/reference/Sitecore%206/IE%20Configuration%20Reference.aspx.

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.

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.
2. In the Content Explorer, double-click any item. Content Editor appears in the Developer Center with the 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. 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.
- **`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 Visual Studio 2008. User interface steps for other languages or versions of Visual Studio may differ.

### 2.4.1 How to Show Visual Studio Solution Explorer

To show Solution Explorer in Visual Studio, in the Web application project, 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 Solution Explorer, and then toggle Show All Files (typically the second button from the left at the top of Solution Explorer).

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

### 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.[4]

**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 Templates list, click ASP.NET Web Application.

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

4. 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 Location field, enter the document root of the Sitecore solution, for example `C:\inetpub\sitecore\MyWebSite\WebSite`.
6. Clear the Create directory for solution checkbox.
7. Accept the default value in the Location field, and then click OK. Visual Studio creates a subdirectory in the directory specified by the Location field. The project appears in Visual Studio.

To move the Visual Studio project:

1. Close Visual Studio.
2. In 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 clear the Create directory for solution checkbox when you created the project, 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.
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.

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 assembly browser dialog appears.
4. In the assembly browser 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. 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, 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. 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. Click the Application tab.
4. In the Assembly name field, enter a name for the assembly for the project to generate, without the `.dll` extension. For example, enter `MyWebSite`.
5. In the Default Namespace field, 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.
2. In Visual Studio Solution Explorer, show all files.
3. In Solution Explorer, expand the directory containing the file.
4. In Solution Explorer, right-click the file, and then click Include In Project.

**Important**
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.

To add Sitecore controls to the Visual Studio toolbox:

1. In Visual Studio, close any open solutions or projects.
2. 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 in the document root of the Web site, and then click the `Sitecore.Kernel.dll` assembly.
5. Click Open, but do not click OK until reading the following information.

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

To limit the controls added to the Toolbox:

1. In the control selection dialog, sort by Assembly Name.
2. Select the checkbox next to any of the Sitecore controls to clear the checkboxes for all of the controls in the Sitecore library.
3. Sort by Namespace.
4. Scroll to the `Sitecore.Web.UI.WebControls` namespace.
5. 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. Right-click in the Visual Studio Toolbox, and then click Add Tab. For the tab name, enter Sitecore.
2. 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 the Visual Studio Web application project, right-click in the Toolbox, and then click Choose Items. The control selection dialog appears.
2. In the control selection dialog, sort by Namespace.
3. Scroll to the `Sitecore.Web.UI.WebControls` namespace.
4. Clear the checkboxes next to each of the Sitecore controls.

-or-

1. In Visual Studio, close any open projects and solutions.
2. Click the View menu, and then click Toolbox.
3. Right-click in the 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 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, hide all files.
3. Navigate to the appropriate line in the relevant .NET code file.
4. To create a breakpoint, click the Debug menu, and then click Toggle Breakpoint, or press F9.
5. Click the Debug menu, and then click Attach to Process, or press CTRL-ALT-P.
6. Select the Show processes from all users and Show processes in all sessions checkboxes.
7. 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).
8. Click Attach.
9. Use a Web client to request an ASP.NET resource that uses the code to debug.

Top stop debugging, in 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 built into Visual Studio, which can have undesirable results.

## 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 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, for Templates, select Web Service.
4. In the Name field, enter a name for the file that will contain the Web service methods, and then click Add.
5. Create Web service methods in the new code file.

### 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 RSS feeds:

1. In the Visual Studio options dialog, expand Environment, and then click Startup.
2. Clear the value in the Start Page news channel field.
3. Clear the Download content every checkbox.

To disable the start page:

1. In the Visual Studio options dialog, expand Environment, and then click Startup.
2. 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 splash screen:

1. Right-click the shortcut you use to start Visual Studio.
2. In the Target property, add the `/nosplash` command line option. 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. Clear the Animate environment tools checkbox.

If you use Jetbrains ReSharper, pressing CTRL-F12 provides update the list of methods and fields above the editing pane.5 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#.

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

2. Clear the Navigation bar checkbox.

To disable change tracking:

1. In the Visual Studio options dialog, click Text Editor.
2. Clear the Track changes checkbox.

To turn off active item tracking:

1. In the Visual Studio options dialog, click Projects and Solutions.
2. Clear the Track Active Item in Solution Explorer checkbox.

To turn off AutoToolboxPopulate:

1. In the Visual Studio options dialog, click Windows Forms Designer.
2. 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. For Start pages in, select the Source View option.

# 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 Open the Device Editor

To open the Device Editor:

1. In the Template Manager or the Content Editor, edit the standard values item or the individual item.
2. Click the Presentation tab.
3. 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.
2. In the Layout drop-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.
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 the Open Properties after this dialog closes checkbox. For more information about rendering properties, see the section Controls.
4. Click Select.

## How to Order Controls

To order controls in layout details:

1. In the Device Editor, click the Controls tab.
2. Click a control to select it.
3. 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.
2. On the Controls tab, click a control to select it
3. With the control selected, click Remove.

## How to Replace a Control

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

1. In the Template manager or Content Editor, in layout details for the standard values or individual item, for the relevant device, click Edit. The Device Editor appears.
2. In the Device Editor, click the Controls tab, then click the rendering, and then click Change. The Select a Rendering dialog appears.
3. In the Select a Rendering dialog, select the new rendering, and then click Select.

## 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, edit the item for which you will reset layout details to those defined in the standard values item of the data template associated with the item.
2. Click the Presentation tab.
3. In the Layout group, click Reset. 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, navigate to the source item from which to copy layout details.
2. Click the Presentation tab.
3. In the Details group, click the Layout command. The layout details dialog appears.
4. In the layout details dialog, under any device, click Copy To. The copy layout details wizard appears.
5. In the copy layout details wizard, in the Target Devices list, select the checkboxes next to the devices for which to copy layout details.
6. In the Target Item tree, click the item to which you will copy layout details.
7. Click Copy. The copy layout details wizard copies 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, navigate to the source item.
2. Show standard fields and raw values.[6]
3. In the source item, in the Layout section, triple-click the value of the Renderings field to select that value, and then press CTRL-C to copy that value to the Windows clipboard.
4. For each target item, navigate to the item. In the Layout section, triple-click the value of the Renderings field in the Layout section to select that value, and press CTRL-V to overwrite that value with the value from the Windows clipboard.
5. Hide the standard template fields and raw field values.

[6] 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 Sitecore's browser-based debugger to determine the presentation components used to service an HTTP request.

**Important**
The Sitecore debugger uses the default device and the publishing target database unless otherwise specified.

**Important**
Publish changes to layout details before debugging.

To determine the presentation components used to render a page:

1. In the Sitecore Debugger, in the Rendering group, select the Information checkbox.
2. In the Trace group, select the Active command.
3. Hover over the information icons (the green triangles) and investigate the controls.
4. Scroll down to Sitecore Trace and look for errors in 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, navigate to `/Sitecore/Layout/Devices`.
2. Insert a device definition item using the `/System/Layout/Device` data template.
3. If the device should have a fallback device, in the device definition item, in the Data section, in the Fallback device field, select the fallback device.
4. Define the criteria that trigger the device.

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

### 3.5.2 How to Define Device Activation Criteria

If the layout engine should activate the device based on a specific user-agent string sent in the headers of HTTP requests, enter that user-agent string in the Browser Agent string in the Detection section of the device definition item. For example, enter `blackberry` to activate the device for all blackberry clients. The user agent string comparison is not case sensitive.

If the layout engine should activate the device based on a specific query string parameter value, enter that value in the Query String field in the Detection section of the device definition item. For example, enter `x=1` to activate the device for all URLs containing the query string parameter `x` with a value of `1`.

If the layout engine should activate the device for all HTTP requests associated with a specific hostname, configure a logical site in `web.config` and set the device attribute.[7]

If .NET logic should activate the device, implement that logic and set `Sitecore.Context.Device` using Visual Studio.[8] Most commonly, this logic belongs in a pipeline processor to replace or follow the default `Sitecore.Pipelines.HttpRequest.DeviceResolver` processor in the `httpRequestBegin` pipeline.

[7] For more information about configuring multiple logical sites, see http://sdn.sitecore.net/Articles/Administration/Configuring%20Multiple%20Sites.aspx.
[8] 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 the Control Properties Dialog
- 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, in the Rendering group, select the Information checkbox.
2. Hover over the information icon (the green triangle) representing the rendering.
3. Click the Output tab. The output of the rendering appears.

## 4.2 Rendering Definition Items

Before you can use a presentation component in Sitecore, you must register the component. Sitecore user interfaces including the Developer Center, the Device Editor, and the Page Editor Design Pane allow you to use only those presentation components registered with Sitecore by creating a definition item. If users do not work with a presentation component in Sitecore, it is not necessary to create a definition item for the component in Sitecore.

**Important**
In the Data section of each rendering definition item used with the Page Editor Design Pane, in the Description field, enter a description of the output of the component, potentially including an image demonstrating the location of the placeholder in a layout or sublayout, or the output of a sublayout or rendering. For components that support caching, include a reminder to set caching options if you do not apply caching options to the presentation component definition item. For cases such as 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 creating a rendering definition item, if necessary, use the Content Editor and Windows file system Explorer to create Sitecore folders and file system directories to contain the definition item and the file the definition item will reference.

**Note**
The term rendering definition item includes both rendering definition items and sublayout definition items.

## 4.3 How to Access the Control Properties Dialog

To access the control 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 layout or sublayout, click the Design tab.
3. In the layout or sublayout, double-click the control. The control 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, and then click Edit. The control properties dialog appears.
2. In the control properties dialog, click the Attributes tab. Unless otherwise specified, create or update all control properties using the Attributes tab.

## 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. 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 in 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 update a rendering definition item, Sitecore does not update layouts and sublayouts to which you have statically bound the rendering. You should also update the layouts and sublayouts that statically bind the rendering.

**Important**
If users use the Developer Center to update layouts and sublayouts, define default caching options in each rendering definition item. These values provide defaults when a user add a rendering to a layout or sublayout.

**Important**
To avoid unnecessary memory consumption, avoid caching the output of renderings used in cached sublayouts.

To configure caching options using the control properties dialog:

1. In the control properties dialog, click the Caching tab.
2. Configure 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 to a control using the control properties dialog, in the Control Properties Dialog, in the Data Source field, enter the full path to the data source item, or click Browse and select an item.

To pass a data source item to a control 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, for the DataSource property, enter the full path to an item.

### 4.5.3 How to Pass Parameters to a Control

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

1. In the control properties dialog, click the Parameters tab.
2. Enter named parameters.

**Important**
To set properties of a Web control, including the `FieldName` property of the FieldRenderer Web control, use the Parameters tab, not the Attributes tab. If needed, use the parameter named `Parameters` to set the `Parameters` property of the Web control.

To pass parameters to a control 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, for the Parameters property, enter named parameters using URL-escaped key=value pairs, separated by ampersand characters ("`&`").

## 4.6 Placeholders

The only property specific to placeholders is the placeholder key.

**Note**
Because a placeholder is a control, Sitecore uses the same dialog to set the properties on a placeholder that it uses to set properties on all controls. If you double-click a placeholder, sublayout, or any type of rendering control, Sitecore shows the same control properties dialog as when you set control properties in layout details. In the Developer Center, the Placeholder field in this dialog is irrelevant; for placeholders, enter the placeholder key using the `key` attribute on the Parameters tab. In layout details, for each control, 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 the Developer Center or Visual Studio, 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.

**Warning**
Do not cache the output of sublayouts and Web controls that respond to ASP.NET events.

**Warning**
To avoid excess memory consumption, do not cache both the output of a rendering, and the output of a sublayout that contains that rendering.

## 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 the FieldRenderer Web control (`Sitecore.Web.UI.WebControls.FieldRenderer`). Define 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. 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 Visual Studio.

In the Path field in the Data section of an XSL rendering definition item, enter path to the XSL rendering file relative to the document root of the Web site, such as `/xsl/mywebsite/myrendering.xslt`.

**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 the 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 presentation component dialog appears.
2. In the new presentation component dialog, in the Categories tree, click Renderings.
3. In the Templates list, click XSLT File, and then click Create.
4. For Name, enter a name for the XSLT rendering, and then click Next. The wizard will use this name for both the XSLT rendering definition item and the XSLT rendering file.
5. In the content tree, click the item that will contain the XSL rendering definition item, and then click Next.
6. 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. At the top of the editing window, ensure the Preview button is selected.
3. In the drop-down list in the previewing pane below the editing pane, 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. 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 Main XSL Template Block

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

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

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

### 4.9.4 The XSL Rendering Boilerplate File

The boilerplate file that the Developer Center uses when creating a new XSL rendering contains the following lines.

```
<?xml version="1.0" encoding="UTF-8"?>
```

XSL rendering files contain XSL code. XSL is a dialect of XML; all XSL files are XML files.

```
<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">
```

The root element of the XML document contained in an XSL file is the `<xsl:stylesheet>` XSL element. In the same way that ASP.NET maps tag prefixes to assemblies, XSL maps namespace identifiers to URLs. The `xsl` namespace exposes the XSL language. Sitecore by default defines the `sc` and `dot` namespaces, which correspond to classes. A `/configuration/sitecore/xslExtension/extension` element in `web.config` provides the signature for each class, effectively mapping a namespace to a URL, and a URL to a .NET assembly. For more information about .NET XSL extensions, see the section Custom XSL Extension Methods.

```
<xsl:output method="html" indent="no" encoding="UTF-8" />
```

This rendering outputs markup using HTML syntax, which does not require closing elements for `<hr>`, `<img>`, `<meta>`, and other elements. For an XHTML site, the value of the `method` attribute should be `xml`.

```
<xsl:param name="lang" select="'en'"/>
<xsl:param name="id" select="''"/>
<xsl:param name="sc item"/>
<xsl:param name="sc_currentitem"/>
```

Sitecore passes several parameters to the rendering:

- **`$lang`**: The context language.
- **`$id`**: The GUID of the data source item for the rendering.
- **`$sc_item`**: The data source item for the rendering.
- **`$sc_currentitem`**: The context item.

```
<xsl:variable name="home"
  select="$sc_item/ancestor-or-self::item[@template='site root']" />
```

This line provides an example of creating a variable that references an item.

```
<xsl:template match="*">
  <xsl:apply-templates select="$sc_item" mode="main"/>
</xsl:template>
```

This code sets the data source of the rendering (`$sc_item`) as the context element and invokes the following XSL template with `mode` attribute `main`.

```
<xsl:template match="*" mode="main">
</xsl:template>
```

Developers generally begin coding by inserting code within this `<xsl:template>` block.

```
</xsl:stylesheet>
```

This line closes the `<xsl:stylesheet>` root element.

## 4.9.5 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.[9]

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:

```
<xsl:template>
  <xsl:choose>
    <xsl:when test="//TODO:logic">
      Variable Value
    </xsl:when>
    <xsl:otherwise>
```

[9] 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.

```
            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.

## 4.9.6 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.[10] 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

[10] 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 is as follows:

```
<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 representing 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">
```

## 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.

## 4.9.7 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.

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

To set Web control properties, in the control properties dialog, click the Attributes tab, and add or update or named properties.

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

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

For the Namespace property, 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. Replace `Namespace.Web.UI.WebControls` with the appropriate namespace to contain the class.
3. Replace `ClassName` with the name of the class.
4. Replace `//TODO: write to output` with logic to write to the `output HtmlTextWriter`.
5. 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.

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 the Renderings node.
3. In the Templates list, click Web Control, and then click Create. The Web control configuration dialog appears.
4. In the Web control configuration dialog, enter Web control properties.
5. Click Test, resolve any issues until the Developer Center successfully finds the Web control, and then click Next.
6. 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 within `/Sitecore/Layout/Renderings` that will contain the Web control definition item.
2. Insert a Web control definition item using the `/System/Layout/Renderings/Webcontrol` data template.
3. 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 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 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 in the `/Sitecore/Layout/Renderings/System` folder, and use the name Method Rendering.

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

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

For the Assembly property, 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 options. To support output caching for method renderings, create a Web control that inherits from the method rendering Web control (`Sitecore.Web.UI.WebControls.Method`). Define the `GetCachingID()` method to return a cache key for the control, for example a string containing the namespace, class, and method name. Use this Web control instead of the default method rendering Web control. Alternatively, create Web controls to invoke methods, and define the `GetCachingID()` method in these 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. 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 Templates list, click Method Rendering, and then click Create.

4. Enter method rendering properties.
5. Click Test, resolve any issues until the Developer Center successfully finds the method, and then click Next.
6. Click the folder 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, navigate to the folder within `/Sitecore/Layout/Renderings` that will contain the method rendering definition item.
2. Insert a method rendering definition item using the `/System/Layout/Renderings/Method Rendering` data template.
3. In the method rendering definition item, enter method rendering properties as appropriate.

## 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**
URL renderings invoke the rendering from the server, not from the client.

### 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 Templates list, click Url Rendering, and then click Create.
4. Enter URL rendering properties.
5. Click Test, resolve any issues with the URL, and then click Next.
6. Click the folder 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, navigate to the appropriate folder within `/Sitecore/Layout/Renderings`.
2. Insert a URL rendering definition item using the `/System/Layout/Renderings/Url Rendering` data template.
3. In the URL definition item, specify URL rendering properties as appropriate.

## 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, in a project-specific folder, insert a rendering settings data template definition item using the `System/Layout/Rendering Parameters/Standard Rendering Parameters` data template as the base template.
2. 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.

3. In the Content Editor, select the rendering definition item for which the properties apply.
4. 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.
5. 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 Templates list, click Layout, and then click Create.
4. In the Name field, enter a name for the layout, and then click Next. The wizard will use this name for both the layout definition item and the layout file.
5. Click the folder that will contain the layout definition item, and then click Next.
6. 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. Create the Web form in the appropriate project-specific 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 appropriate project-specific folder within `/Sitecore/Layout/Layouts`, for example `/Sitecore/Layout/Layouts/MyWebSite`.
3. Insert a layout definition item using the `/System/Layout/Layout` data template. For the name of the item, enter the name of the Web form file, without the `.aspx` extension, for example `MyWebLayout`.
4. In the layout definition item, ignore any exception on the Grid Designer tab, and click the Content tab.
5. 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**
Do not attempt to register an existing Web form as a Sitecore layout using the Layout command template that appears in insert options for the `/Sitecore/Layout/Layouts` folder. The Layout command template attempts to create the layout file, and generates an error if that file already exists.

## 5.2 Create a Sublayout

You can use the Developer Center to create a sublayout, or register a Web user controls created in Visual Studio or another text editor 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 Templates list, click Sublayout, and then Click Create.
4. For Name, enter a name for the sublayout, and then click Next. The Developer Center will use this name for both the sublayout definition item and the sublayout file.
5. Click the item that will contain the sublayout definition item, and then click Next.
6. Click the directory that will contain the sublayout file.
7. Select the Create Associated C# Code Files checkbox to create a code file.
8. Click Create. The new 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 Content Editor, navigate to the project-specific folder under `/Sitecore/Layout/Sublayouts` that will contain the sublayout, for example `/Sitecore/Layout/Sublayouts/MyWebSite`.
2. Insert a sublayout definition item using the `/System/Layout/Sublayout` data template. For the name of the item, enter the name of the Web user control file, without the `.ascx` extension, for example `MySublayout`.
3. In the sublayout definition item, ignore any exception on the Grid Designer tab, and click the Content tab.
4. 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**
Do not attempt to register an existing Web form as a Sitecore sublayout using the Sublayout command template that appears under the Insert menu when you right-click the `/Sitecore/Layout/Sublayouts` folder. The Sublayout command template attempts to create the sublayout file, and generates an error if that file exists.

## 5.3 Add a Control to a Layout or Sublayout

This section provides procedures for adding placeholders and statically binding sublayouts and renderings to placeholders in layouts and sublayouts.

**Note**
Developers generally dynamically bind sublayouts to placeholders, and rarely statically bind sublayouts to layouts or other sublayouts. If several layouts or sublayouts always share a single sublayout, such as several layouts that use the same header sublayout, the developer might statically bind the sublayout.

**Important**
Never allow a sublayout to bind statically or dynamically to another instance of itself or to a control which is a descendant of the sublayout, as this 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 layout or sublayout, click the Design tab.
3. Click the View menu, and then click Toolbox.
4. Drag the control from the Toolbox onto the layout or sublayout.
5. In the layout or sublayout, double-click the control. The control properties dialog appears.
6. In the control 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. Edit the layout or sublayout that will contain the control.
3. In the layout or sublayout, click the Design tab.
4. Drag the control from the toolbox onto the layout or sublayout, or insert the control markup. For a placeholder, use the following markup template:

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

   For a sublayout, use the following markup template:

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

   For an XSL rendering, use the following markup template:

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

   For a FieldRenderer Web control, use the following markup template:

```
<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 template:

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

For a method rendering, use the following markup template:

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

For a URL rendering, use the following markup template:

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

5. Right-click the control, and then click Properties. The Visual Studio Properties pallet appears.
6. In the Properties pallet, apply control properties.

**Note**
To support Sitecore output caching for statically bound sublayouts, use the Sitecore sublayout Web control instead of invoking the Web user control directly.

**Important**
You cannot pass parameters to a sublayout bound statically using named control properties. Instead, use the parameters property of the Sitecore sublayout Web control.

## 5.4 Add Code-Beside to a Layout or Sublayout

When you create a layout or sublayout in the Developer Center, the Developer Center does not create a code-beside file by default. You can add a code-beside file to an existing layout or sublayout in one of two ways: by copying the text of, then deleting the existing layout or sublayout file, recreating that file in Visual Studio, and then pasting the text, or by creating 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 Visual Studio Web application project, open the layout or sublayout. Note the exact name and location of the layout or sublayout file.
2. In the layout or sublayout, click the Source tab.
3. 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. Press CTRL-C to copy the selected text to the Windows clipboard.
5. In Visual Studio Solution Explorer, delete the layout or sublayout.
6. Use Visual Studio Solution Explorer to add a new Web Form (layout) or Web User Control (sublayout) using the original layout or sublayout file name in the directory that contained it.
7. In the new layout or sublayout file, click the Source tab.
8. Select all of the text except for the `Page` or `Control` directive at the top of the file.
9. Press CTRL-V to overwrite the existing text with the text in the Windows clipboard.

### 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 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, press F2 to rename the file, then CTRL-C to copy the file name to the Windows clipboard, then ESC to cancel the rename operation.

2. In 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, in the Name field, 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 the class, enter an appropriate namespace and class name.
5. Open the layout or sublayout.
6. Click the Source tab.

7. 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" %>
```

8. For the `inherits` attribute, enter the namespace and class name. For the `codebehind` attribute, enter the name of the code-behind file.
9. 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 the Visual Studio Web application project, in Solution Explorer, right-click the `.aspx` or `.ascx` file, and then click Include In Project.
2. 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.