Sitecore CMS 6.1

Presentation Component Reference

A Conceptual Overview for CMS Administrators, Architects, and Developers

Table of Contents

Chapter 1

Introduction

This document provides a conceptual overview of the Sitecore layout engine for CMS architects, developers, and administrators.[1] Use this document to identify and define the presentation components required to implement a Sitecore solution in order to minimize development, maintenance, and administration costs.

This document contains the following chapters:

- Chapter 1 — Introduction
- Chapter 2 — Presentation Components
- Chapter 3 — Request Processing
- Chapter 4 — Developer Center and Microsoft Visual Studio
- Chapter 5 — Output Caching
- Chapter 6 — Choosing Presentation Technology

[1] For instructions to use the features described in this document, see the Presentation Component Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx. For information about implementing XSL renderings, see the Presentation Component XSL Reference at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20XSL%20Reference.aspx. For more information about implementing .NET rendering components, see the Presentation Component API Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20API%20Cookbook.aspx.

Chapter 2

Presentation Components

This chapter begins with an overview of the Sitecore layout engine, and then continues to provide details about the different types of presentation components employed.

This chapter contains the following sections:

- Layout Engine Overview
- Layouts (ASP.NET .aspx Web Forms)
- Sublayouts (ASP.NET .ascx Web User Controls)
- Renderings
- Placeholders
- XML Layouts, XML Controls, XML Dialogs, and XML Forms

2.1 Layout Engine Overview

The Sitecore layout engine extends the ASP.NET Web application server to separate content from presentation until the Web client requests a resource. The layout engine dynamically merges content in the database with code in files according to layout details defined in the requested content item, automatically accounting for the context of the user including language, device, and access rights.

ASP.NET represents pages as hierarchies of .NET objects called controls. Each control is responsible for rendering a different component on the page. Sitecore adds facilities beyond those provided by the underlying ASP.NET constructs. These facilities include:

- The ability to easily invoke XSL transformations in addition to .NET logic.
- The ability to bind controls to placeholders dynamically and declaratively during page requests using layout details.
- The ability to cache the output of individual presentation components by various criteria.

Layout details in each content item in the database (or the standard values of the data template associated with the item) control the presentation components that the layout engine uses to service HTTP requests for that item from different types of devices. Sitecore assembles the page response from this hierarchy of presentation components, each of which can invoke any API or access any aspect of the CMS, including all content, metadata, item relations, and configuration settings.

Presentation components may generate different output depending on the user's context, such as the user's profile including security authorization, the requested language, ASP.NET page and control events, and other criteria. Logic in the layout engine and individual presentation components applies the user's session to process content in the CMS database or other systems.

Sitecore provides the browser-based Developer Center application for registering and working with presentation components. Developers often manage presentation components using Microsoft Visual Studio and source code management systems.

Important
Sitecore URLs correspond to content items in the database that reference presentation components implemented as files on disk. Sitecore URLs do not correspond directly to files on disk, but to data points that reference ASP.NET and XSL presentation components implemented in files on disk.

2.2 Layouts (ASP.NET .aspx Web Forms)

Layouts define the outermost markup common to the greatest number of responses to HTTP requests, such as an outer structure common to most or all pages of a Web site. Developers use layouts:

- To define the outermost markup superstructures shared to the greatest number of pages.
- To define the outermost markup layer for individual pages or other features, such as a layout for a site's home page that differs from all other pages.
- To reuse the same markup superstructure for any number of page views, dynamically binding other presentation components to enclosed placeholders as defined by layout details for the requested item.
- To apply different presentation components when different types of devices request a content item, such as formatting content differently for Web browsers and mobile devices.

2.2.1 Layout Implementation

ASP.NET uses `.aspx` files, known as Web forms, to service HTTP requests. Sitecore layouts are ASP.NET Web forms registered with the content management system, including both a layout definition item and an `.aspx` file, sometimes with additional `.dll` and `.cs` files implementing code-behind or code-beside functionality. Layouts are `.aspx` files that are registered with Sitecore using layout definition items.

Layouts definition items beneath the `/Sitecore/Layout/Layouts` item that are based on the `System/Layout/Layout` data template. The Path field in each layout definition item references the path to an `.aspx` file relative to the document root of the Web site.

Each layout file contains markup representing a hierarchy of literal and server controls. Literal controls represent markup elements, such as HTML tags. Server controls represent ASP.NET classes that generate output dynamically. The layout engine parses the layout, writing the literal controls to the HTTP response stream and invoking the server controls to write to that stream. Server controls generate output dynamically and can respond to page events. Server controls provided by Sitecore include:

- Sublayouts
- XSL renderings
- Web controls
- Placeholders
- URL renderings
- Method renderings
- Web parts

Developers bind some server controls, including placeholder controls, to the layout at design time. The layout engine replaces these statically-bound controls with their output each time it services an HTTP request with that layout. The layout engine binds additional server controls to placeholder controls in the layout dynamically, at runtime according to the layout details of the requested item, replacing each placeholder with the output of the specified controls. For more information about placeholders, see the section Placeholders. For more information about layout details, see the section Layout Details.

Note
Not every request processed by the Web server activates a layout. Some versions of IIS do not use ASP.NET to process all HTTP requests, serving files from disk without providing Sitecore with an opportunity to intercept the request.[2] When IIS uses ASP.NET, some requests activate media or other handlers, which do not apply layout details. Based on configuration, Sitecore ignores some requests, allowing IIS to handle those requests without Sitecore as standalone ASP.NET pages.

2.2.2 Layout Usage

Each logical Web site usually involves at least one layout per device. For example, a Web site that supports Web browsers and mobile devices can use one layout for Web browsers and another for mobile devices. Layouts are generally the most reusable presentation component. Some Sitecore solutions use a single layout for all pages.

Placeholders in layouts and sublayouts represent regions where developers use layout details to cause the layout engine to invoke different presentation components dynamically to service different types of requests.

Each HTTP request activates at most a single layout. IIS does not activate Sitecore to service all requests, and Sitecore ignores other requests, allowing ASP.NET to process those requests as it would by default. Sitecore and ASP.NET process some requests using other ASP.NET handlers that do not involve Sitecore layouts.

Tip
Minimize the number of layouts required by dynamically binding presentation components using placeholders and layout details.

Note
Unlike sublayouts, layouts rarely involve code-behind or code-beside. Instead, global logic often belongs in `global.asax`, a pipeline, an event handler, or elsewhere, while application logic in application components such as sublayouts.

[2] For information about configuring IIS to process additional requests using ASP.NET, see the guide to Dynamic Links at http://sdn.sitecore.net/Reference/Sitecore%206/Dynamic%20Links.aspx.

2.3 Sublayouts (ASP.NET .ascx Web User Controls)

Sublayouts define markup substructures intended for use within a layout or a sublayout nested in and bound statically or dynamically to a layout or another sublayout. Developers use sublayouts:

- To contain markup substructures shared to multiple pages, where that substructure may function as a superstructure for further nested components.
- To reuse an outer shell defined in a common layout with different internal dimensions defined in different sublayouts.
- To nest components dynamically using placeholders in sublayouts.
- To implement ASP.NET applications.
- To render output on a page, using a sublayout as a rendering.
- To contain reusable groups of controls, such as a header sublayout that developers bind to multiple layouts, similar to include files but with dynamic features.

2.3.1 Sublayout Implementation

In ASP.NET, `.aspx` files can contain `.ascx` files, known as Web user controls. Web user controls populate portions of the response to an HTTP request. Sitecore sublayouts are ASP.NET Web user controls registered with the by inserting a sublayout definition item. A sublayout is both the sublayout definition item and the `.ascx` file, plus any `.dll`, `.cs`, or other resources supporting the Web user control.

Sublayout definition items, stored under the `/Sitecore/Layout/Sublayouts` item, use the `System/Layout/Sublayout` data template. The Path field in each sublayout definition item references the path to an `.ascx` file relative to the document root of the Web site, plus any `.cs`, `.dll`, or other files related to the Web form.

Just like layouts, each sublayout contains a hierarchy of literal and dynamic server controls. Developers bind some server controls to sublayouts statically and others to placeholders in the sublayout dynamically using layout details.

The layout engine invokes zero or more sublayouts to service portions of each request that it processes using a layout. The layout engine dynamically binds presentation components to placeholders in the layout and nested placeholders in sublayouts according to layout details. For more information about layout details, see the section Layout Details.

2.3.2 Sublayout Usage

While developers can statically bind sublayouts by adding the sublayout to a layout or sublayout, developers more frequently bind sublayouts to placeholders in layouts and sublayouts dynamically using layout details. Sublayouts may in turn contain nested placeholders, supporting nesting of presentation components to any level.

Developers commonly use sublayouts to implement ASP.NET applications. Sublayouts are therefore more likely than layouts to have code-behind or code-beside. The output of sublayouts, which may represent application data and content generated from multiple distinct presentation components, is often less reusable than the output of individual renderings. Sublayouts are therefore less likely than other types of renderings to support output caching.

2.4 Renderings

Renderings are individual presentation components that function as building blocks for published Web sites. Developers use renderings:

- To present content.
- To present data from external systems.
- To perform back-end logic with no visual components, such as logging requests for Web analytics or other purposes.

Note
In most cases, the term rendering can refer to a sublayout.

2.4.1 Rendering Implementation

The layout engine uses Web controls to invoke all types of presentation components other than layouts, including sublayouts, XSL renderings, method renderings, URL renderings, and Web parts. A rendering includes a rendering definition item that specifies file(s) containing the code that implements that rendering.

Note
XSL renderings include a rendering definition item, the `Sitecore.Web.UI.WebControls.XslFile` (`<sc:xslfile>`) Web control that invokes the transformation, and the `.xslt` file that contains the transformation code. Sublayouts include a sublayout definition item, the `Sitecore.Web.UI.WebControls.Sublayout` (`<sc:sublayout>`) Web control that invokes the sublayout, the `.ascx` Web user control file that implements the sublayout, and possibly code-behind in a `.dll`, `.cs`, or other file(s).

2.4.2 Rendering Usage

Developers bind some renderings to layouts and sublayouts statically at design time, causing the layout engine to invoke those renderings every time it processes that layout or sublayout. Developers bind other renderings to placeholders in layouts and sublayouts dynamically at runtime using layout details.

2.4.3 Rendering Types

Sitecore supports a variety of rendering technologies. For information about which technology to use for specific tasks, see Chapter 6, ‘Choosing Presentation Technology’.

Sublayouts as Renderings

You can use a sublayout as a rendering, to generate output on a page.

XSL Renderings

XSL renderings output the results of XSL transformations. The source for the XSL transformation is an XML representation of a Sitecore database, though an XSL rendering may use the `document()` function to access external XML resources.

Web Control Renderings

All ASP.NET pages are hierarchies of literal and dynamic server controls. ASP.NET Web controls are ASP.NET page elements implemented as classes that eventually inherit from `System.Web.UI.Control` in the .NET framework, typically through `System.Web.UI.WebControls.WebControl`. ASP.NET Web controls generate output dynamically and respond to page events, typically by ovedrriding the `Render()` method to generate output.

Sitecore Web control renderings are ASP.NET controls that inherit from the Sitecore Web control base class `Sitecore.Web.UI.WebControl`, which inherits from `System.Web.UI.WebControls.WebControl`. Sitecore Web control renderings override the `DoRender()` method to generate output.

In addition to the features inherited from the ASP.NET base control class, Sitecore Web control renderings support:

- Passing a data source item to the control.
- Dynamic binding to placeholders.
- Rendered output caching as described in Chapter 5, Output Caching.

The Sitecore Web control base class provides additional convenience methods and properties for Sitecore developers.

Note
Web controls that do not use the Sitecore Web control features described previously may inherit directly from a .NET system class rather than inheriting from the Sitecore Web control base class.

Tip
The layout engine can dynamically bind Web controls that inherit from the Sitecore Web control base class to Sitecore placeholders, but cannot dynamically bind Web controls that inherit directly from the ASP.NET Web control base class. If needed, you can statically bind Web controls that inherit from the ASP.NET base class to a sublayout, and dynamically bind that sublayout to a placeholder.

Method Renderings

Method renderings write the string returned from a .NET method to the output stream.

URL Renderings

URL renderings request a URL and write the response to the output stream. Unlike iframes in which the Web client requests a URL, with URL renderings, the server requests the URL while generating a page response, and embeds the result in that response. If the response from the URL contains an HTML `<body>` element, the layout engine only outputs the contents of that element, not the `<body>` element itself or any elements outside the body element, such as `<html>`, `<head>`, or `<form>`.

Web Part Renderings

The optional Web parts framework allows you to use of Web parts as renderings.[3]

[3] For more information about using Web parts with Sitecore, see the Sitecore Web Parts Framework at http://sdn.sitecore.net/Resources/Free%20Modules/Web%20Part%20Framework.aspx.

2.4.4 Rendering Properties

Properties, or field values in each rendering definition item, control how Sitecore uses that rendering.

Warning
When you use the Developer Center or the Grid Designer to add a rendering to a layout or sublayout, Sitecore copies properties from the definition item to the new reference to the rendering in the layout or sublayout file. If you change rendering properties in the rendering definition item, Sitecore does not update the corresponding parameters in all layouts and sublayouts that reference that rendering.

Note
Developers control rendering properties. Parameters Template Rendering Parameters allow users to control renderings.

Tip
To let the user select different groups of default rendering properties including Rendering Parameters, insert multiple rendering definition items for a single rendering.

The Description Rendering Property

Sitecore displays Description property when the user hovers over the rendering in user interfaces such as the Design pane of the Page Editor.[4]

The Parameters Template Rendering Property

Sitecore uses the data template specified by the Parameters Template property to define the fields that appear in the Control Properties dialog when a user defines Rendering Parameters.[5]

Like any other data template, a rendering settings data template defines fields. A rendering settings data template inherits from the `System/Layout/Rendering Parameters/Standard Rendering Parameters` base template that defines rendering parameters common to all types of renderings. If you do not specify a rendering parameters template for a rendering, then Sitecore uses the `System/Layout/Rendering Parameters/Standard Rendering Parameters` data template as the parameters template.

Unlike other data templates, no items are associated with the rendering settings data template. Sitecore stores values entered into the Control Properties dialog in the layout details of the item. For more information about layout details, see the section Layout Details.

Warning
When you update a rendering settings data template, such as by renaming or removing a field, Sitecore does not automatically update layout details, which may reference that field by name.

Important
Rendering settings data templates do not support standard values. Instead, use The Parameters Template Rendering Property.

[4] For more information about the Page Editor, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Client%20Configuration%20Cookbook.aspx.
[5] For more information about the Control Properties dialog, see the Presentation Component Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx.

The Open Properties After Add Rendering Property

If the value of the Open Properties After Add rendering property is Yes, then Sitecore displays the Control Properties user interface after the user adds that rendering to layout details.[6]

The Open Properties After Add rendering property is a Tristate field in rendering definition items that affects the Open Properties after this dialog closes checkbox in the Select a Rendering dialog. The following table describes the effect of different values of the Open Properties After Add rendering property:

Open Properties After Add	Open Properties after this dialog closes
Default	Clear, enabled.
No	Clear, disabled.
Yes	Selected, enabled.

The Customize Page Rendering Property

The Customize Page property is a legacy replaced by parameters The Parameters Template Rendering Property.

Warning
Use The Parameters Template Rendering Property instead of the Customize Page rendering property. If you have implemented the customize page feature, consider moving to parameters templates.

The Placeholder Rendering Property

The Placeholder rendering property defines the default placeholder key for the rendering. Sitecore uses the Placeholder rendering property if a user does not specify a placeholder in layout details or if a developer does not specify a placeholder when adding the rendering to layout details at runtime using conditional rendering.

The Parameters Rendering Property

The Parameters rendering property allows you to enter default values for Rendering Parameters.

Note
Define the Parameters rendering property in rendering definition items using URL query string parameter encoding. To determine a value to store in the Parameters rendering property, bind the rendering statically to a temporary layout or sublayout, then define properties for the rendering using the Developer Center or the Grid Designer, then view the source of the layout or sublayout, and copy the value of the `parameters` attribute of the control to the Parameters field in the rendering definition item.

Caching Rendering Properties

The caching rendering properties provide default output caching options for the rendering. For more information about caching options, see the Chapter 5, Output Caching.

Sitecore copies caching options from the rendering definition item to the control when you bind a rendering statically to a layout or sublayout using the Developer Center or the Grid Designer. When you

[6] For more information about the Control Properties dialog, see the Presentation Component Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx.

bind a rendering to a placeholder dynamically using layout details, unless you specify caching options in layout details, the layout engine dynamically applies caching options from the rendering definition item.

Type-Specific Rendering Properties

In addition to the general rendering properties described in this section, various types of renderings expose properties relevant to that type. For example, XSL renderings expose the Path property, which specifies the path to an `.xslt` file.[7]

2.4.5 Rendering Parameters

Users can specify any number of parameters for each rendering. Developers use rendering parameters:

- To avoid hard-coding content, configuration, and other data in renderings.
- To reuse renderings with different configurations.
- To allow users to control rendering features.

You can apply rendering properties and parameters in three places: in the rendering definition item, in the layout or sublayout when you bind the rendering statically, or in layout details when you bind the rendering dynamically to a placeholder.

Important
When you bind a rendering statically using Visual Studio or any text editor, define rendering parameters in the layout or sublayout, even if the rendering definition item defines the corresponding properties or parameters.

Tip
Define rendering properties in the rendering definition item. Define rendering parameters where you bind renderings statically to layouts and sublayouts and also when you bind renderings dynamically to placeholders using layout details. Define default rendering parameters as described in the section The Parameters Rendering Property.

The Placeholder Rendering Parameter

When you bind a rendering to a placeholder dynamically using layout details, you must specify the key of the placeholder. If the placeholder exists in the layout or any of the sublayouts bound to that layout or one of its nested sublayouts before the layout engine processes that rendering reference in layout details, and the user has read access to that rendering, and conditional rendering rules do not prevent the rendering from running, Sitecore binds the rendering to the specified placeholder.

Note
The Placeholder field in the Sitecore user interface is only relevant when binding a rendering field to a placeholder.

The Data Source Rendering Parameter

Each rendering can accept a data source item from which to begin processing. Developers use rendering data sources:

[7] For more information about rendering properties specific to different types of renderings, see the Presentation Component Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx.

- To avoid hard-coding item paths or GUIDs, such as CSS classes or IDs.
- To reuse renderings with different data.
- To specify data for the rendering to process.

The context item, which corresponds to the URL requested by the Web client, is the default data source item for all renderings for which you do not specify a data source.

You can specify a data source when you bind a rendering to a placeholder dynamically using layout details, or when you bind a rendering to a layout or sublayout statically.

The Caching Rendering Parameters

In addition to caching rendering properties, which provide default caching options, you can define cache rendering parameters when you bind a rendering dynamically to a placeholder using layout details, and when you bind a rendering statically. For more information about output caching, see the Chapter 5, Output Caching.

Note
Caching rendering parameters defined in the layout, sublayout, or layout details override caching rendering properties defined in the rendering definition item. For more information about caching rendering properties, see the section Caching Rendering Properties.

The Personalization Rendering Parameter

The Personalization rendering parameter allows the user to select conditional rendering rules to apply to the rendering.

The Tests Rendering Parameter

The Tests rendering parameter specifies multivariate tests to apply when invoking the rendering.

The Additional Parameters Rendering Parameter

The Additional Parameters rendering parameter allows the user to enter rendering parameters as a list of keys with corresponding values. You can use the Additional Parameters rendering parameter for parameters with no corresponding field in the parameters template specified in the rendering definition item. For more information about parameters templates, see the section The Parameters Template Rendering Property.

Parameters Template Rendering Parameters

In addition to the default rendering parameters, you can use parameters templates to define custom rendering parameters. Sitecore displays the fields defined in the parameters template in the Control Properties user interface for the rendering.

For XSL renderings, custom rendering parameters correspond to parameters created using the `<xsl:param>` element within the root `<xsl:stylesheet>` element. For .NET renderings, custom rendering parameters correspond to properties of the .NET object identified by the rendering definition item.

2.4.6 The FieldRenderer Web Control

Sitecore provides the FieldRenderer Web control to output a single field value, automatically expanding dynamic links and providing inline editing controls in the Page Editor.[8] Developers use the FieldRenderer Web control to retrieve and format a single field value.

The FieldRenderer Web control exposes the `DataSource` parameter, which controls the item from which Sitecore retrieves the field value. If you do not specify the `DataSource` parameter, then the layout engine processes the field from the context item.

The FieldRenderer Web control also supports the following parameters:

Parameter	Function
`After`	Text to output after the field value (inside the closing `EnclosingTag` if supplied both).
`Before`	Text to output before the field value (inside the opening `EnclosingTag` if supplied both).
`DisableWebEditing`	True to disable inline editing for the field.
`EnclosingTag`	Markup element to wrap field value (for example, `div`).
`FieldName`	Name of field to process.

The `Sitecore.Web.UI.WebControls.FieldRenderer` class in the `Sitecore.Kernel` assembly provides the implementation of the FieldRenderer Web control. The `/Sitecore/Layout/Renderings/System/FieldRenderer` Web control rendering definition item references this class, making it easy to drag this control onto a layout or sublayout in the Developer Center.

[8] For more information about dynamic links, see the guide to Dynamic Links at http://sdn.sitecore.net/Reference/Sitecore%206/Dynamic%20Links.aspx.

2.5 Placeholders

Sitecore placeholders are ASP.NET controls that define named regions of layouts and sublayouts to which other controls bind dynamically at runtime according to layout details.[9] Developers use placeholders:

- To represent regions of a reusable layout or sublayout in which different components execute for different requests.
- To invoke different presentation components in different regions of a markup structure without duplicating that markup structure.

2.5.1 Placeholder Implementation

Placeholders associate locations in a layout or sublayout with a name known as a placeholder key. The layout engine dynamically substitutes placeholders with the sublayouts and renderings associated with that key in the layout details of the requested item. Consistency leads to usability, and usability leads to return visitors. Developers achieve consistency through content and code reuse. Placeholders maximize consistency while minimizing development and maintenance through reuse of presentation components across various types of content and even different logical sites.

A Sitecore placeholder is a Web control that exposes properties including `Key`. Unlike other Web controls, you always bind the placeholder Web control statically to a layout or sublayout; you never bind a placeholder to a placeholder dynamically using layout details.

Each layout and sublayout can contain any number of placeholders. Placeholders support nesting to any level; a sublayout may bind to a placeholder in a sublayout that binds to a placeholder in a layout. Layout details allow any number of presentation components to bind to each placeholder. If the layout details associate multiple presentation components with the same placeholder key, the layout engine binds those components to the placeholder in the order specified in layout details.

Note
The `value` of the `/configuration/sitecore/settings/setting` element in `web.config` with name `LayoutPageEvent` controls which ASP.NET page event causes the layout engine to apply layout details. Developers can choose to bind presentation components to placeholders during the `PreInit` event, the `Init` event, or the `Load` event.

2.5.2 Placeholder Keys

Each placeholder has a textual key. Each presentation component specified in layout details indicates the key of the placeholder. These placeholder key references in layout details instruct the layout engine which presentation components to bind dynamically to each placeholder when generating page views, and in what order to bind components to each placeholder.

Placeholder keys must be unique within all of the presentation components referenced in the layout details of any device for an individual item. It is invalid for both a layout and a sublayout used for a single device in the presentation details for a single item to contain multiple placeholders with a common key, such as a nested sublayout containing a placeholder with the same key as a placeholder in the layout.

[9] Do not confuse Sitecore placeholders with content placeholders used in ASP.NET master pages. All uses of the term placeholder in Sitecore documentation refer to Sitecore placeholders unless otherwise specified.

Placeholders in common regions defined in multiple components never used together in a single page commonly share a single placeholder key. For example, two sublayouts never used together in a single page view might each contain a placeholder with a common key.

Layout details can reference placeholders by key or by fully qualified key. A fully qualified placeholder key indicates the location of the placeholder in the component nesting hierarchy. For example, if a sublayout containing a placeholder with key `B` binds to a placeholder with key `A` in a layout, the fully qualified key of the nested placeholder is `/A/B`. The **Design** pane of Page Editor always uses fully qualified placeholder keys, but users may enter unqualified placeholder keys in layout details.

2.5.3 Placeholder Settings

Placeholder settings provide properties for placeholders, such as to control which sublayouts and renderings users can bind to each placeholder.[10]

2.5.4 Placeholder Usage

Developers use placeholders to represent regions of reusable layouts and sublayouts in which different components execute for different content items. Developers use layout details to cause the layout engine to bind different presentation components to the placeholders for different types of items.

Each rendering component can generate output dynamically. Placeholders add the ability to execute different rendering components for different items that share a common layout.

Tip
To minimize administration of layout details, use placeholders only when necessary. Instead of dynamically binding renderings to placeholders in layout details, statically bind presentation components to layouts and sublayouts whenever possible.

[10] For more information about placeholder settings, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206/Client%20Configuration%20Cookbook.aspx.

2.6 XML Layouts, XML Controls, XML Dialogs, and XML Forms

Sitecore implements CMS user interfaces with XML layouts, XML controls, XML dialogs, and XML forms. This document does not describe these technologies because developers do not generally use them to develop published Web sites, only CMS user interface components.

Chapter 3

Request Processing

This chapter describes layers features that Sitecore adds to the ASP.NET page lifecycle.

This chapter contains the following sections:

- The Sitecore Layout Engine
- Devices
- Layout Details

3.1 The Sitecore Layout Engine

The Sitecore layout engine enhances the ASP.NET page lifecycle:

- With ASP.NET, URLs correspond to files within the document root of the physical directory represented by the document root of the Web site. With the Sitecore layout engine, URLs correspond to items in a database, where the path to that item is relative to the home item of the logical Web site activated by the URL. Instead of assembling the page from the components specified an `.aspx` file, the layout engine assemblies the page from the components specified in the layout details of the item with the path that corresponds to the requested URL.
- The HTTP module defines a context object indicating the user, language, requested item, device, and other contextual information.

Mapping URLs to items allows Sitecore to invoke different presentation components when different types of devices request the item, and to provide conditional rendering, analytics, and other features.

Web applications respond to HTTP requests from Web browsers, RSS readers, mobile, and other types of devices. The Sitecore layout engine uses properties of each HTTP request to determine how to assemble the response. The layout engine invokes the presentation components specified in the layout details of the item requested by the Web client. These ASP.NET and XSL presentation components assemble the HTTP response by accessing the CMS database and any other resources available to ASP.NET and XSL.

3.1.1 The Context Item

The context item is the default source of data during the request lifecycle. The layout engine identifies the content item in the database based on the path in the requested URL.

The context item is the default data source item for most operations associated with the page request, such as determining layout details to apply. The context item is the default data source for renderings for which you do not specify a data source.

For example, under the default configuration, if a Web client requests the path `/hr/jobs.aspx`, the layout engine sets the context item to the `/Sitecore/Content/Home/hr/jobs` item in the context database.

3.2 Devices

Devices represent different types of Web clients that connect to the Internet and place HTTP requests to other connected devices, such as Web servers. Each device can represent various types of Web clients with varying markup requirements.

The layout engine applies the presentation components specified for the context device in the layout details of the context item. Developers use devices to format a single item using different collections of presentation components for various types of Web clients.

3.2.1 Device Implementation

The layout engine determines the context device for each HTTP request:

- By applying any custom .NET logic.
- By comparing URL query string parameters against registered devices.
- By comparing the Web client user-agent against registered devices.
- From the properties of the context site.

If the request does not activate a specific device, the layout engine assumes the Default device as the context device. The default device typically represents a Web browser.

Fallback Device

You can associate each device with a fallback device. If the context item does not contain layout details for the context device, then the layout engine applies the layout details for the fallback device

Note
If defined, layout details for the fallback device in the context item override layout details for the context device in the standard values of the data template associated with the item. If the context item does not contain layout details for the context device, but does contain layout details for its fallback device, the layout engine applies the layout details in the context item for the fallback device without evaluating layout details for the context device in the standard values of the data template associated with the item.

3.2.2 Device Usage

Sitecore provides two devices by default:

- The Default device, which typically represents a Web browser.
- The Print device, which represents a page prepared to be sent to a printer.

By default, the layout engine activates the Print device if the URL query string includes the parameter `p` with a value of `1` (`p=1`). If the layout engine does not activate the Print device, it activates the Default device.

You can register any number of devices to represent additional types of clients, or other criteria requiring the layout engine to format content differently than it should for other devices. Additional devices may include but are not limited to the following:

- RSS readers.

- Mobile devices.
- Specific Web browsers.
- Silverlight or Flash, such as to generate XML in a format for a Microsoft Silverlight or Adobe Flash component to consume.
- Multiple logical Web sites.

Note
You must define criteria to activate each device.

3.3 Layout Details

Layout details specify the reusable presentation components that the layout engine should invoke to service requests for items from different types of Web clients. Use layout details:

- To control the layout, sublayouts, and renderings that the layout engine invokes to service HTTP requests for individual items or types of items.
- To declaratively reuse presentation components to service requests for multiple items.
- To define multiple variant presentations of an item for different types of devices.

3.3.1 Layout Details Implementation

ASP.NET Web applications map URLs in HTTP requests to files within the document root of the IIS Web site. For example, when IIS processes an HTTP request for `/hr/jobs.aspx`, it invokes ASP.NET to process the file `jobs.aspx` in the `/hr` directory within the document root of the IIS Web site.

The Sitecore layout engine maps URLs to content items in a database. HTTP requests activate Sitecore items in the database, not files. Layout details in each content item specify the layout, sublayout, and renderings to process when different types of devices request each item.

The standard template inherited by most other data templates defines a field to contain layout details. These layout details for each device in each content item indicate which layout to apply, and which sublayouts and renderings to populate each placeholder in the layout and any nested sublayouts.

Layout details instruct the layout engine to dynamically populate the same placeholders in common layouts and sublayouts with different components for different items or types of items. Each presentation component can generate output dynamically using data in a Sitecore database, Sitecore APIs, or any APIs or other resources available to ASP.NET or XSL.

Layout details separate data from presentation until runtime, providing content and presentation component reuse, flexibility in administration, simplified global user interface changes such as rebranding, support for distinct presentation for sub-sites, and other user interface administration requirements.

Tip
To minimize administration, rather than defining layout details in each content item, define layout details in the standard values of each data template.

3.3.2 Layout Details vs. ASP.NET Content and Master Pages

ASP.NET supports content pages (`.aspx` files) that reference master pages (`.master` files) controlling presentation of the content.[11] ASP.NET master pages contain controls that ASP.NET applies to multiple content pages.

Sitecore enhances the ASP.NET page generation process by providing abstract content storage and declarative layout details stored in a database instead of the file system. While layout files may be ASP.NET content pages that reference master pages, Sitecore developers generally avoid master pages and content pages due to the greater flexibility and reusability provided by declarative layout details.

Sitecore items are logically similar to ASP.NET content pages in that they contain content and reference presentation components, but with much greater flexibility than that provided by ASP.NET content pages.

[11] For more information about ASP.NET maser pages and content pages, see http://msdn2.microsoft.com/en-us/library/wtxbf3hh.aspx.

Instead of referencing a single master page, a content item may reference different layouts, sublayouts, and renderings to invoke when the different types of devices request the item. Unlike ASP.NET content pages, you can translate items into any number of languages, and process items using any number of layouts, sublayouts and renderings for different devices.

Sitecore layouts are similar to ASP.NET master pages in that they contain controls applied to a number of content items. Unlike master pages that allow only one layer of nesting, layout details support declarative nesting of presentation components to any level.

ASP.NET master pages involve a page-based architecture, which can present challenges when developing navigation and reusing content. The Sitecore layout engine provides a data-driven architecture that facilitates navigation and reuse.

Chapter 4

Developer Center and Microsoft Visual Studio

This chapter describes the advantages and disadvantages of using Sitecore's browser-based Developer Center application to maintain presentation components in contrast to using Microsoft Visual Studio. This chapter also describes release management considerations specific to Sitecore presentation components.

This chapter contains the following sections:

- Developer Center vs. Visual Studio
- Presentation Component Definition Items

4.1 Developer Center vs. Visual Studio

The Developer Center provides allows you to edit layouts, sublayouts, and XSL renderings using only a Web browser. You can also edit presentation components in offline file editors such as Visual Studio.

The Developer Center has some advantages over Visual Studio:

- The Developer Center does not require client license or client installation.
- The Developer Center is less intimidating and easier to learn than Visual Studio.
- The Developer Center is available through the Sitecore desktop.
- The Developer Center provides easy access to Sitecore's browser-based debugger.

Visual Studio has some advantages over Developer Center. Visual Studio provides:

- A single development environment for all types of files including ASP.NET, XSL, CSS, JavaScript, and other resources.
- IntelliSense, syntax completion, automatic code indentation, error underlining, and other integrated development environment features.
- The Visual Studio debugger for .NET code.
- Source code management tool integration.

Note
The ASP.NET 2.0 Web Application project model available in Visual Studio 2005 Service Pack 1 and Visual Studio 2008 is appropriate for most Sitecore solutions.[12]

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

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

4.2 Presentation Component Definition Items

All presentation components involve files on disk, such as `.ascx`, `.aspx`, `.xslt` code, or .NET assemblies. Other presentation elements, such as CSS, JavaScript, media, and other files referenced by presentation components consist of files on disk, which may or may not have corresponding definition items.

Layouts, sublayouts, XSL renderings, and other types of items consist of a definition item containing a field that contains the path to file on disk that implements presentation logic. Web control rendering definition items and method rendering definition items store the name of a class and the .NET assembly containing that class.

Certain operations on definition items have predictable consequences that can be unexpected for developers unfamiliar with Sitecore. Moving, duplicating, renaming, or deleting a definition item does not update the field that contains the file location information. A duplicate of a definition item references the same file as the original definition item.

Other than media stored as files instead of in the database, developers typically manage file assets using a source code management system with release management techniques rather than using CMS versioning and publishing.

Important
Be sure to deploy both the file and the definition item when moving presentation components from development through test to production.

Chapter 5

Output Caching

This chapter describes how the layout engine caches the output of different presentation components to maximize performance and throughput.

This chapter contains the following sections:

- Rendered Output Caching Options
- Rendered Output Caching Implementation

5.1 Rendered Output Caching Options

Each presentation component may generate different output under different conditions. For example, a footer rendering might generate the same output for all pages, while a breadcrumb rendering may generate different output for different URLs, and a navigation rendering might generate different output for different users based on their access rights to content items.

The layout engine can cache the output of each sublayout and rendering used by each page view. Developers use rendered output caching to improve performance by not executing sublayouts and renderings under different conditions, instead retrieving output generated previously by that component under the same conditions.

While output caching does not eliminate the need for data structure and code optimization, avoiding code execution can increase performance significantly, especially in high-volume solutions.

Page caching, such as by using the OutputCache directing in an ASP.NET Web form, can consume excess memory a component generates the same output for numerous page views. Page caching does not support dynamic features. Sitecore component output caching allows the output of each component to vary by a number of criteria as described in the following sections, caching only when appropriate.

Important
Caching is crucial to overall solution performance. The quickest and easiest way to increase the throughput, and hence capacity, of a Sitecore solution is to optimize output caching configuration.

Important
Do not cache the output of components that respond to ASP.NET page events without understanding the implications.

Important
Do not confuse Sitecore rendered output caching with ASP.NET page and fragment caching as implemented with the OutputCache directive in Web forms and Web user controls. Developers should not use ASP.NET page and fragment caching with Sitecore content, or must clear the ASP.NET cache when required, such as after Sitecore publishing operations. In Sitecore documentation, the term caching refers to Sitecore rendered output caching unless otherwise specified.

Important
Developers must override the `GetCachingID()` method in Web controls in order to support output caching. This method generally returns an identifier for the rendering, such as the namespace and class name of the Web control.

5.2 Rendered Output Caching Implementation

By default, the layout engine executes each presentation component for each HTTP request. Developers must select caching criteria for each use of each presentation component that requires output caching.

The layout engine manages a separate output cache for each logical site. Caching automatically varies by site.

Each output cache logically consists of a list of any number of key-value pairs. Each cache key is a unique string identifying a presentation component and various caching criteria. The value in the cache corresponding to that cache key is the output of that component under those criteria. Multiple invocations of a single presentation component may generate multiple entries in the list referencing output generated under different conditions, each with a different cache key.

The cache key automatically includes the context language and a presentation component identifier, such as the ID of the rendering definition item or the path to an XSL rendering file. Caching automatically varies by component and language.

Note
Caching may vary by multiple criteria. For example, a developer may choose to vary the output caching of a presentation component by both data source and other *VaryBy* properties.

Setting the *VaryBy* properties described in the following sections to true, adds corresponding tokens to the cache key, causing caching to vary by those properties. When the layout engine evaluates a presentation component configured to cache output, it retrieves output from the cache if an entry with the corresponding key exists in the cache.

If the developer has not configured the component to cache output, or the cache does not contain a corresponding entry, the layout engine invokes the component. If the developer has configured the component to cache, the layout engine adds an entry with the corresponding key to the cache.

Caching the output of each sublayout and rendering by the fewest criteria possible minimizes memory usage and the number of times the system must execute each component.

Note
By default, publishing clears output caches.

5.2.1 Which Cache Settings Apply?

Sitecore allows developers to define output cache criteria in three places:

- In the Caching section of the sublayout and rendering definition item.
- In the properties of the control where a developer statically binds a presentation component to a layout or sublayout.
- On the Caching tab when a developer binds the presentation component to a placeholder in layout details.

The layout engine uses cache criteria defined in the Caching section of the definition item in only two cases:

- When a developer statically binds a rendering to a layout or sublayout using the Developer Center or the Grid Designer, Sitecore copies caching properties from the definition item to the control (a static reference to the rendering).

- The layout engine uses caching settings in the definition item if layout details do not specify caching criteria for components dynamically bound to placeholders.

You must explicitly define cache settings wherever you bind a rendering to a layout or sublayout. When you dynamically bind a rendering to a placeholder using layout details, cache settings explicitly defined in layout details override cache settings defined in the rendering definition item. Cache settings defined in the definition item apply only when no caching settings exist on the Caching tab in layout details.

Note
Caching options defined in the Caching section of the sublayout or rendering definition item provide default caching criteria for users who define layout details.

5.2.2 Output Caching Properties

Cacheable presentation components support the following caching properties. All caching properties default to false.

Cacheable

The *Cacheable* property of each use of a presentation component controls whether or not the layout engine caches the output of that component. If the *Cacheable* property is false, the layout engine invokes the component each time it processes the component reference. The layout engine never caches or retrieves the output of the component from cache, regardless of any of the *VaryBy* properties defined in the following sections.

If the *Cacheable* property is true and no *VaryBy* properties are true, the layout engine invokes the component on its first use for each logical site for each language, but retrieves that cached output for all subsequent uses of that component for that logical site and language. If the *Cacheable* property is true and one or more *VaryBy* properties are true, those *VaryBy* properties control whether or not the layout engine invokes the component or retrieves cached output generated previously under the same *VaryBy* conditions.

Developers set the *Cacheable* attribute:

- To False, for any sublayouts and renderings for which the layout engine should not cache output.
- To True, for any sublayouts and renderings for which the layout engine should cache output.
- To True, with no true *VaryBy* properties for components for which output does not vary by any criteria other than logical site and language.
- To True, with one or more true *VaryBy* properties for components that generate different output under the specified conditions.

Note
If the *Cacheable* property of a presentation component is false, *VaryBy* properties have no effect. The layout engine never caches the output of components for which the *Cacheable* property is false or undefined.

Important
The developer must define caching properties for each use of each cacheable component.

VaryByData

The *VaryByData* property controls whether or not output caching varies based on the data source of the presentation component.

Developers set the *VaryByData* property:

- To False, for components that do not generate different output when used with different data sources.
- To True, for components that generate different output when used with different data sources.

VaryByDevice

The *VaryByDevice* property controls whether or not caching varies based on the name of the context device.

Developers set the *VaryByDevice* property:

- To False, for components that do not generate different output when used with different devices.
- To True, for components that generate different output when used with different devices.

VaryByLogin

The *VaryByLogin* property controls whether or not output caching varies based on whether or not the user has authenticated.

Developers set the *VaryByLogin* property:

- To False, for components that do not generate different output for authenticated than for unauthenticated visitors.
- To True, for components that generate different output for authenticated than for unauthenticated visitors.

Note
For caching configuration involving *VaryByLogin*, the layout engine treats all anonymous users as a single authenticated user.

VaryByParm

The *VaryByParm* property controls whether or not output caching varies based on rendering parameters passed to the presentation component.

Developers set the *VaryByParm* property:

- To False, for components that do not generate different output when passed different rendering parameters.
- To True, for components that generate different output when passed different parameters.

Note
Solutions built with earlier versions of Sitecore may have used the token *VaryByParam* instead of *VaryByParm*. Update any uses of *VaryByParam* to *VaryByParm*.

VaryByQueryString

The *VaryByQueryString* property controls whether or not output caching varies based on query string parameters passed in the URL.

Developers set the *VaryByQueryString* property:

- To True, for components that generate different output when supplied different query string parameters.
- To False, for components that do not generate different output when supplied different query string parameters.

Note
Do not confuse *VaryByParm* with *VaryByQueryString*. *VaryByParm* causes output caching to vary based on rendering parameter values passed by the developer. *VaryByQueryString* causes output caching to vary based on parameters passed in the URL query string.

VaryByUser

The *VaryByUser* property controls whether or not output caching varies by the domain and username of the context user.

Developers set the *VaryByUser* property:

- To False, for components that do not generate different output for different users.
- To True, for components that generate different output for different users, when the number of active users between publishing operations is relatively small.

Note
For caching configuration involving *VaryByUser*, the layout engine treats all anonymous users as a single authenticated user.

Note
To avoid excess memory consumption, avoid *VaryByUser* except in solutions with relatively small numbers of users or supported by sufficient hardware resources.

Note
Do not confuse *VaryByUser* with *VaryByLogin*. *VaryByLogin* causes the presentation component to generate different output depending on whether or not a user has authenticated, differentiating anonymous users from authenticated users. *VaryByUser* causes the presentation component to generate different output for each user.

Chapter 6

Choosing Presentation Technology

This chapter provides guidance for choosing a technology to implement each presentation component.

This chapter contains the following sections:

- General Presentation Technology Considerations
- Specific Presentation Technology Considerations

6.1 General Presentation Technology Considerations

Developers choose a technology to implement each presentation component. In some cases, requirements of the component dictate or restrict the choice of presentation technologies. For example:

- Layouts represent markup superstructures shared to numerous pages.
- Placeholders represent regions of layouts and sublayouts to which the layout engine dynamically binds various sublayouts and renderings according to layout details.
- Existing Web forms and Web user controls convert most easily to sublayouts.
- Only layouts and sublayouts support placeholders.
- Web control renderings can easily wrap or replace existing Web controls.
- Method renderings can reference existing .NET methods.
- Web part renderings can reference existing Web parts.
- URL renderings embed content retrieved from another URL.
- XML layouts, XML controls, XML dialogs, and XML forms are appropriate for CMS user interfaces.

For other presentation components, developers choose between implementing a sublayout, an XSL rendering or a Web control rendering. In general, XSL renderings are appropriate for components containing mostly markup, while sublayouts and Web control renderings are appropriate for presentation components containing significant logic.

6.2 Specific Presentation Technology Considerations

The following sections describe specific advantages and disadvantages of the various presentation component technologies used to render portions of a page.

6.2.1 Sublayout Considerations

Sublayouts provide all of the features of ASP.NET Web user controls. Sublayouts separate design (the `.ascx` file) from logic (the optional code-behind or code-beside file). Sublayouts have access to the Sitecore context, the Sitecore database and .NET APIs, as well as any resources and APIs available to ASP.NET. Sublayouts support nested ASP.NET controls including Sitecore placeholders. Developers can step through compiled sublayout code using the Visual Studio debugger.

6.2.2 XSL Rendering Considerations

With only a little knowledge of XSL and XPath syntax, XSL can be a powerful language for generating markup, similar to HTML but with logic to generate output dynamically. XSL is an open standard defined by the W3C. For developers fluent in the technology, XSL can be efficient and elegant for a variety of presentation tasks.

XSL is perfectly suited for generating markup, especially by reformatting XML into HTML or XHTML. XSL is often appropriate for retrieving and formatting field values, such as in main content body renderings, as well as recursive functions, such as site maps and breadcrumbs. XSL can be especially useful in prototyping, and developers can convert XSL renderings to one of the .NET technologies if needed.

XSL uses text files editable through any text editor or even Sitecore browser-based user interfaces, which do not require a compiler or restart ASP.NET when updated.

Sitecore XSL extension controls and functions automatically support inline editing of field values in the CMS database.

The preview frame beneath the editing pane in Developer Center allows the developer to select a data source item and view the output of XSL renderings in real time while editing the code.

XSL and XPath syntax, as well as the lack of common programming features available to XSL code, result in XSL being ill suited for complex logic. The declarative programming model is unfamiliar to many developers. Complex XPath and other statements in XSL code can be difficult to maintain. XSL is generally not suited to working with multiple data sources, especially resources other than XML.

While developers often edit XSL in integrated development environments such as Visual Studio, such offline XSL editors cannot access the XML representations of Sitecore databases or invoke .NET XSL extensions. Developers cannot debug XSL renderings using Visual Studio, though XSL renderings can write to the trace visible in Sitecore's browser-based debugger. XSL renderings do not provide compile-time error detection, only runtime exception management.

XSL renderings execute source code, which must exist in all environments including production. Dynamically interpreted XSL may never perform as well as native .NET code. Cache the output of all renderings by the fewest criteria possible, especially expensive XSL renderings.

Whether or not a project uses XSL, Sitecore developers must be familiar with ASP.NET. Because a developer could accomplish anything in .NET that they could accomplish in XSL, XSL is an optional technology requiring an additional developer skill set to implement and support. Avoid implementing the same logic in both .NET and XSL code.

XSL renderings can invoke .NET logic through extensions, allowing presentation control through flexible XSL markup with logic in compiled .NET code. While XSL transformation performance may never equal that of a native .NET component, the advantages of XSL for formatting may outweigh the performance differential, especially for components that support output caching.

XSL renderings cannot contain nested placeholders or ASP.NET controls.

6.2.3 Web Control Considerations

Web control renderings support all of the features of ASP.NET Web controls. Web controls have access to the Sitecore context, the Sitecore database and .NET APIs, as well as any resources and APIs available to .NET. These features are a superset of those available to XSL renderings. Developers can step through compiled Web control code using the Visual Studio debugger.

Web controls do not separate design from presentation using the ASP.NET code-behind and code-beside models used by sublayouts, and are therefore appropriate for components that generate markup completely dynamically. Web controls cannot contain placeholders.

6.2.4 Method Rendering Considerations

Method renderings are very simple and efficient, but do not support a data source, rendering parameters, or output caching. In general, wrap methods with Web control renderings rather than implementing method renderings in order to support caching.