Sitecore CMS 7.0 or later

Presentation Component Reference

A Conceptual Overview for CMS Administrators, Architects, and Developers

Table of Contents

Chapter 1

Introduction

This document describes the presentation components used by the Sitecore layout engine, including layouts, sublayouts, renderings, and placeholders. CMS architects, developers, and administrators should read this document before implementing Sitecore solutions.[1] Use this document to identify presentation components and technologies to minimize development, maintenance, and administration costs.

This document first describes the types of presentation components, including layouts, sublayouts, and renderings. This document then explains how the layout engine assembles presentation components to service HTTP requests, including devices and layout details. This document then describes how to cache the output of presentation components, and then provides considerations for choosing between presentation technologies for each component.

This document contains the following chapters:

- Introduction
- Presentation Components
- Request Processing
- Output Caching
- Choosing Presentation Technology

[1] For instructions to use the features described in this document, see the Presentation Component Cookbook.
For information about implementing XSL renderings, see the Presentation Component XSL Reference.
For more information about implementing .NET rendering components, see the Presentation Component API Cookbook.

Chapter 2

Presentation Components

After providing an overview of the Sitecore layout engine, this document describes the types of presentation components that it uses, including layouts, sublayouts, renderings, and placeholders. This chapter concludes with a description of the XML presentation components used by the Sitecore user interfaces.

This chapter contains the following sections:

- Layout Engine Overview
- Layouts (ASP.NET .aspx Web Forms)
- Sublayouts (ASP.NET .ascx Web User Controls)
- Renderings
- Placeholders
- Sitecore User Interface Presentation Components

2.1 Layout Engine Overview

The Sitecore layout engine extends the ASP.NET web application server to merge content with presentation logic dynamically when web clients request resources. The layout engine dynamically merges content in the database with code in files according to layout details defined in the context item, which corresponds to the URL requested by the web client. The layout engine determines the context language, device type, and other criteria from the HTTP request. Presentation components generate different output based on the Sitecore context. For more information about the context item, see the section The Context Item

The ASP.NET web application server represents pages as hierarchies of literal and server controls. Literal controls represent markup elements, such as HTML tags. Server controls represent ASP.NET classes that generate output dynamically. Each control is responsible for rendering a different component on the page. ASP.NET assembles responses to HTTP requests by assembling the literal controls and invoking the server controls to write to the response stream. Server controls generate output dynamically and can respond to page events.

The Sitecore layout engine adds facilities beyond those provided by ASP.NET, including:

- The layout engine makes it easy to invoke XSL transformations in addition to .NET logic.
- The layout engine allows you to use bind controls to placeholders dynamically and declaratively through a browser-based user interface.
- The layout engine provides the ability to cache the output of each presentation components by different criteria.
- The layout engine provides conditional rendering, allowing you to use logic to affect the controls used to service a page request at runtime.

The layout engine services HTTP requests with the presentation components specified for the context device in layout details for the context item, or in the standard values of the data template associated with the context item if that item does not define layout details. Sitecore assembles the page response from the hierarchy of presentation components specified in layout details for the context device. Each presentation component can invoke any .NET API including Sitecore APIs, and can access any content, metadata, item relations, and configuration settings including Sitecore repositories.

Each presentation component can generate different output depending on the context of the user, such as whether the user has authenticated, the profile of the user including security authorization, the requested language. Like any ASP.NET control, presentation components can respond to events, such as when the user clicks the component.

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
In many web solutions, HTTP requests correspond to files on disk. Sitecore HTTP requests correspond to items in a Sitecore database. Each item contains layout details. Layout details instruct Sitecore which presentation components to apply. Developers manage presentation components as files on disk, such as ASP.NET web forms (`.aspx`), XSL transformation files (`.xslt`), and .NET assemblies (`.dll`). For more information about layout details, see the section Layout Details.

Important
Presentation components include both a definition item in Sitecore and zero or more files on disk. When you copy presentation components from one Sitecore instance to another, include any updated definition items and files, and include assemblies rather than code files when possible.

2.2 Layouts (ASP.NET .aspx Web Forms)

Layouts define the outermost reusable markup superstructures common to the greatest number of pages. Developers use layouts:

- 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 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 format content differently to service HTTP requests from different types of devices, such as web browsers, printers, and mobile devices.

2.2.1 Layout Implementation

Sitecore layouts are reusable ASP.NET web forms (`.aspx`) that you register with Sitecore. ASP.NET can use web forms to service HTTP requests. Sitecore services HTTP request by invoking one of the layouts associated with the requested item. Sitecore uses properties of the request and the requested item to determine which layout to invoke.

Layouts include a definition item in Sitecore and an `.aspx` file on the file system. Some layouts include supporting files, such as C# code (`.cs`). Some projects compile code files into .NET assemblies (`.dll`).

Note
In addition to C#, Sitecore supports all languages that ASP.NET supports.

Note
Developers typically manage file assets using a source code management system rather than Sitecore and use release management techniques instead of publishing.

Sitecore manages layouts definition items beneath the `/Sitecore/Layout/Layouts` item in the content tree. Layout definition items use the `System/Layout/Layout` data template. The **Path** field in the **Data** section of each layout definition item references the path to an `.aspx` file relative to the document root of the website.

Using Microsoft **Visual Studio** or the **Developer Center,** you bind controls that the layout always uses to the layout statically, at design time. Each time the layout engine services an HTTP request with that layout, it replaces each control in the layout file (`.aspx`) with the markup generated by invoking that control.

The layout engine determines additional server controls to bind dynamically to placeholders controls in the layout. The layout engine replaces each placeholder in the layout file (.aspx) 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
Sitecore solutions do not use layouts to process every HTTP request. The IIS web serves services some HTTP requests by serving static files from disk. Some requests activate media or other ASP.NET handlers that do not use layout details or layouts. You can configure the `value` attribute of the `/configuration/sitecore/settings/setting` element in `web.config` with name `IgnoreUrlPrefixes` to prevent Sitecore from processing specific requests, causing ASP.NET to process the request without Sitecore.

For information about configuring IIS to use ASP.NET to process additional requests, see the manual *Sitecore Dynamic Links.*

2.2.2 Layout Usage

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

Note
Layouts only involve code when logic applies at the page level. Global logic belongs in another facility such as the global application (`global.asax`), a pipeline processor, or an event handler. Application logic belongs in individual presentation components, such as sublayouts. For more information about sublayouts, see the section Sublayouts as Renderings.

Tip
To minimize the number of layouts required, dynamically bind presentation components using placeholders and layout details.

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

Sublayouts define markup structures use within a layout or nested within another sublayout. Developers use sublayouts:

- To render content on a page.
- To retrieve data from external systems.
- To contain markup substructures shared to multiple pages, where that substructure may function as a superstructure for further nested components.
- To reuse the markup superstructure defined in a layout with different internal dimensions defined in different sublayouts.
- To deeply nest components dynamically using placeholders in sublayouts.
- To implement ASP.NET applications as page components rather than entire pages.
- To contain reusable groups of controls bound to multiple layouts.

2.3.1 Sublayout Implementation

Sublayouts are ASP.NET web user controls that you register with Sitecore. Each HTTP request that activates a layout can invoke zero or more sublayouts and other presentation components to populate that layout. You can bind sublayouts statically to a layout or another sublayout, or dynamically to a placeholder in a layout or nested sublayout using layout details. For more information about layout details, see the section Layout Details.

Sublayouts include a definition item in Sitecore and an `.ascx` file on the file system. Some sublayouts include supporting files, such as C# code (`.cs`). Some projects compile code files into .NET assemblies (`.dll`).

Sitecore manages sublayouts definition items beneath the `/Sitecore/Layout/Sublayouts` item in the content tree. Layout definition items use the `System/Layout/Sublayout` data template. The **Path** field in the **Data** section of each layout definition item references the path to an `.ascx` file relative to the document root of the website.

Each sublayout contains markup representing a hierarchy of literals and dynamic server controls. Developers bind some server controls to sublayouts statically and others to placeholder controls in the sublayout dynamically according to layout details. For more information about placeholders, see the section Placeholders. For more information about layout details, see the section Layout Details.

2.3.2 Sublayout Usage

Sublayouts are the most flexible presentation component. You can use sublayouts to embed markup fragments in a layout or another sublayout, to implement web applications (forms), to create reusable groups of controls, to generate markup dynamically, to invoke APIs that do not generate any markup, or for any other purpose.

Note
Sublayouts are more likely than layouts to include code files.

2.4 Renderings

Renderings are individual presentation components registered with Sitecore that function as building blocks for published websites. Developers use renderings:

- To render content on a page.
- To retrieve data from external systems.
- To perform back-end logic with no visual components, such as logging requests for web analytics or other purposes.

2.4.1 Rendering Implementation

The layout engine uses ASP.NET server controls for all types of presentation components other than layouts. Each rendering includes a rendering definition item that specifies file(s) containing the code that implements that rendering, including parameters to pass to the rendering.

Sitecore supports several types of server controls:

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

Placeholders do not render output, but support dynamic binding of other types of presentation components. For more information about sublayouts, see the section *Sublayouts (ASP.NET .ascx Web User Controls)*. For more information about other types of renderings, see the section Rendering Types.

Note
Method renderings and URL renderings include a rendering definition item containing parameters and a .NET assembly (`.dll` file) installed by Sitecore. Web controls include a rendering definition item and the .NET assembly (`.dll` file) containing the web control class. XSL renderings include a rendering definition item, the .NET assembly installed by Sitecore containing the web control that invokes the transformation, and the `.xslt` file that contains the transformation code. Sublayouts include a sublayout definition item, a .NET assembly installed by Sitecore containing the web control that invokes the sublayout, the `.ascx` web user control file that implements the sublayout, and can include code in one or more `.cs`, `.dll`, or other files.

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. For more information about layout details, see the section 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 5, Choosing Presentation Technology.

Sublayouts as Renderings

You can use a sublayout to generate output on a page. Unless otherwise specified, the term *rendering* includes sublayouts.

XSL Renderings

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

For more information about XSL, see http://www.w3.org/Style/XSL/.

Note
XSL transformation occurs on the web server, not on the web client.

Web Control Renderings

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 `System.Web.UI.Control`, web controls that inherit from `Sitecore.Web.UI.WebControl` support:

- Passing a data source item to the control.
- Dynamic binding to placeholders.
- Rendered output caching.
- 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 `Sitecore.Web.UI.WebControl`.

Tip
The layout engine can dynamically bind web controls that inherit from `Sitecore.Web.UI.WebControl` to Sitecore placeholders. You can statically bind an existing web control that inherits from an ASP.NET base class to a sublayout, and dynamically bind that sublayout to a placeholder.

Method Renderings

When ASP.NET processes a method renderings, it invokes a .NET method that accepts no parameters and writes the string that the method returns to the output stream.

URL Renderings

When ASP.NET processes a URL rendering, it invokes a URL and writes the response to the output stream. If the response 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>`.

Note
Unlike the HTML `<iframe>` element, in which the web client requests URLs, the server requests the specified by a URL rendering.

Web Part Renderings

The optional web parts framework allows you to use of web parts as renderings.

For more information about using web parts with Sitecore, see the Sitecore Web Parts Framework.

2.4.4 Rendering Properties

Rendering properties control how Sitecore uses each rendering. Rendering definition items contain default rendering properties.

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. Rendering parameters allow users to control renderings. For more information about rendering parameters, see the section Parameters Template Rendering Parameters.

Tip
To allow CMS users to select different groups of default rendering properties including rendering parameters, you can insert multiple rendering definition items that supply different rendering properties and parameters to a single piece of rendering code.

The Description Rendering Property

Sitecore displays **Description** property from the **Editor Options** section of the rendering definition item when the user hovers over the rendering in **Page Editor**, design mode.

For more information about the Page Editor, see the manual *Client Configuration Cookbook.*

Rendering Settings Data Templates and the Parameters Template Rendering Property

You can use rendering settings data templates to define the **Control Properties** dialog, providing custom forms for CMS users to enter rendering parameters.

Sitecore uses the data template specified by the **Parameters Template** rendering property in the **Editor Options** section of the rendering definition item to define the fields that appear in the **Control Properties** dialog when a user defines rendering parameters. For more information about rendering parameters, see the section Rendering Parameters.

For more information about the Control Properties dialog box, see the *Presentation Component Cookbook.*

Rendering settings data templates define a structure consisting of one or more data template sections. Each section contains one or more data template fields. The **Control Properties** dialog displays a data entry interface based on the rendering settings data template.

For more information about data templates, see the manual *Data Definition Reference*.

Rendering settings data templates inherit from the `System/Layout/Rendering Parameters/Standard Rendering Parameters` data template. The

`System/Layout/Rendering Parameters/Standard Rendering Parameters` data template 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 for that data template.

You do not create items using the rendering settings data template. Sitecore generates the **Control Properties** dialog based on the rendering settings data template associated with the data template associated with the current item, and stores the values entered in the layout details of that 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 can reference that field by name.

Important
Rendering settings data templates do not support standard values. Instead, use the rendering property named **Parameters**. For more information about the rendering property named **Parameters**, see the section The Parameters Rendering Property.

The Open Properties After Add Rendering Property

If the value of the **Open Properties After Add** rendering property in the **Editor Options** section of the rendering definition item is **Yes**, then Sitecore displays the **Control Properties** user interface after the user adds the rendering to layout details. For more information about layout details, see the section Layout Details.

For more information about the Control Properties dialog box, see the manual *Presentation Component Cookbook*.

The **Open Properties After Add** rendering property 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	Enabled and not selected.
No	Disabled and not selected
Yes	Enabled and selected.

The Customize Page Rendering Property

The **Customize Page** property in the **Editor Options** section of rendering definition items is a legacy replaced by rendering settings data templates.

Warning
Use the **Parameters Template** rendering property rather than 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 in the **Data** section of a rendering definition item specifies the default placeholder key for the rendering. Sitecore uses the **Placeholder** rendering property as the placeholder key if a user does not specify a placeholder key in layout details or if a developer does not specify a placeholder key when adding the rendering to layout details at runtime using conditional rendering. For more information about conditional rendering, see the section Conditional Rendering.

Note
You can use conditional rendering to change the placeholder associated with a rendering at runtime.

The Parameters Rendering Property

The **Parameters** rendering property in the **Data** section of a rendering definition item specifies default values for rendering parameters. For more information about rendering parameters, see the section 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 then 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 4, 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 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.

Note
You can set caching rendering properties dynamically with conditional rendering. For more information about conditional rendering, see the section Conditional Rendering.

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 of rendering.

For more information about rendering properties specific to different types of renderings, see the manual *Presentation Component Cookbook*.

2.4.5 Rendering Parameters

Rendering parameters control how a rendering operates. 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 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.

Note
You can use conditional rendering to change rendering parameters at runtime. For more information about conditional rendering, see the section Conditional Rendering.

Important
When you bind a rendering statically to a layout or sublayout using Microsoft **Visual Studio**, define rendering parameters in the layout or sublayout, even if the rendering definition item defines the corresponding properties and parameters.

Tip
Define rendering properties in the rendering definition item to control the Sitecore user interface. Define rendering parameters where you bind renderings statically to layouts and sublayouts to control how renderings function, and 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

The Placeholder rendering parameter specifies the default key for the placeholder.

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 that the layout engine has processed before processing the rendering, and if the user has read access to that rendering, and if conditional rendering rules do not prevent the rendering from running, then Sitecore dynamically binds the rendering to the placeholder with the specified key. For more information about conditional rendering, see the section Conditional Rendering.

Note
In Sitecore user interfaces including the rendering properties dialog, **Placeholder** is only relevant when binding a rendering 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:

- 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.
- To pass search queries to and parse them in IQueryable Expressions.

  This feature was introduced in Sitecore 7.0. Instead of passing in just one item, you can pass in a query that could evaluate to zero or to many items.

If you do not pass a data source to a rendering, then the context item is the default data source for the rendering. For more information about the context item, see the section The Context Item.

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 *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. For more information about conditional rendering, see the section Conditional Rendering.

The Tests Rendering Parameter

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

For more information about multivariate tests, see the manual *Analytics Configuration Reference*.

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 Rendering Settings Data Templates and 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. Developers use the FieldRenderer web control to retrieve and format a single field value.

For more information about dynamic links, see the manual *Sitecore Dynamic Links*.

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. For more information about the context item, see the section 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`	Set to *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.

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.

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.

Developers use placeholders to:

- Represent regions of a reusable layout or sublayout in which different components invoke for different requests.
- 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.

A class in a .NET assembly (`.dll` file) installed by Sitecore contains the placeholder web control. You do not need to insert any placeholder definition items in the content tree. You can control the renderings that users can bind to placeholders by inserting placeholder settings definition items.

For more information about placeholder settings, see the manual *Client Configuration Cookbook*.

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.

It is preferable that placeholder keys are unique within all of the presentation components referenced in the layout details of any device for an individual item. If both a layout and a sublayout used for a single device in the presentation details for a single item 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, placeholders may not work as expected.

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`. Page Editor, design mode 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.

Important
Sitecore strongly recommends that you use placeholder settings as this will improve performance. If you do not use placeholder settings, every control will try to perform an XPath lookup of the content tree. If the content tree is very large, the performance of the Page Editor will be seriously degraded.

For more information about placeholder settings, see the manual *Client Configuration Cookbook*.

2.5.4 Placeholder Usage

Developers use placeholders to represent regions of reusable layouts and sublayouts to contain different presentation components under different conditions, such as when you use the same layout to process multiple items. Developers use layout details to cause the layout engine to bind different presentation components to the placeholders.

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.

2.6 Sitecore User Interface Presentation Components

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 websites, 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 begins by describing the Sitecore layout engine, which processes requests for Sitecore items. This chapter then explains the concepts of devices and layout details, which allow the layout engine to apply different presentation controls to items depending on properties of the incoming HTTP request.

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 in the following ways:

- The HTTP module defines a context object indicating the user, language, requested item, device, and other contextual information.
- Instead of assembling the page from the components specified in a web form (`.aspx` file), the layout engine assemblies page from the components specified in layout details.

Web applications respond to HTTP requests from various types of web clients including browsers, RSS readers, mobile, and other devices. The Sitecore layout engine uses properties of each HTTP request to determine what type of device requested the item, and therefore how to assemble the response. The layout engine invokes the presentation components specified for the context device in layout details of the context item to assemble an HTTP response. For more information about layout details, see the section Layout Details. For more information about the context item, see the section The Context Item.

By mapping URLs to items in a database instead of files on a file system, Sitecore can dynamically invoke different presentation components to service requests for a single item under different conditions, such as to format content differently when different types of devices request an item. The layout engine also provides conditional rendering, analytics, and other features.

For more information about conditional rendering, see the section *Conditional Rendering*.

For more information about analytics, see the manual *Analytics Configuration Reference*.

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 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 more information about the data source of a rendering, see the section The Data Source Rendering Parameter.

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 represents a different type of web client. Each device can have unique 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 the context item using different collections of presentation components for various types of web clients. For more information about the context item, see the section The Context Item.

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 an HTTP request does not activate a specific device, then the layout engine assumes the **Default** device as the context device. The **Default** device typically represents a web browser.

Fallback Device

You can associate a fallback device with each 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. For more information about the context item, see the section The Context Item.

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, then 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 typically represents a web browser.
- The **Print** device represents 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, then it activates the **Default** device.

You can register any number of custom devices to represent additional types of web clients, or for other purposes that require the layout engine to use different presentation components under different conditions. Additional devices may include but are not limited to the following:

- RSS readers.
- Mobile devices.
- Specific web browsers.

- XML, such as for a Microsoft Silverlight or Adobe Flash component to consume.
- Multiple presentations of a single content item, such as different marketing brands.

Important
Define criteria to activate each custom device.

3.3 Layout Details

Layout details specify reusable presentation components for the layout engine to invoke to service requests for items from different types of devices. 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 reuse presentation components declaratively to service requests for multiple items.
- To define multiple presentations of an item for different types of web clients.

3.3.1 Layout Details Implementation

Layout details in the context item specify the layouts and renderings to invoke when different types of devices request each item. For more information about devices, see the section Devices.

Layout details for each device in each 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 populate the same placeholders in common layouts and sublayouts dynamically 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.

The standard template inherited by most other data templates defines a field to contain layout details.

For more information about the standard template and standard values, see the manual *Data Definition Reference*.

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 Deltas

Layout deltas allow items to inherit partial layout details from a cloned item or standard values from the data template.

When you update layout details on an item that inherits layout details the field that contains layout details stores only the differences from the inherited layout details. Changes to layout details in the base item dynamically apply to items that contain layout deltas.

Layout details are cumulative. A layout delta in a clone applies to a layout delta in the cloned item that applies to standard values.

In Sitecore CMS 6.3 and Earlier

Define layout details on the item template, standard values item.

To change layout details:

- Developers — use the Content Editor and edit the layout details in standard items on the template or customize the layout details on individual content items.
- Business users — Use the Page Editor, Design Pane. The option to *Save as standard layout* means save these values on the standard values item of the template. They can also save the changed layout on the current item only.

When a developer changes the layout on a template standard item this change is inherited by all items that are based on this template.

However, items with their own custom layout details do not inherit values from the standard values item and will be unaffected by layout changes made on the standard values item of the template.

In Sitecore CMS 6.4 and Later

Define layout details on the item template, standard values item.

To change layout details:

- Developers - use the Content Editor and edit the layout details in standard items on the template or customize the layout details on individual content items.
- Business users – Use the Page Editor in Design mode – When they save changes to layout details these changes are saved to the current item only and not to the item template standard values.
  If a Business user wants to reset the item's layout details on the template standard value, they click the Reset button. Developers have more options.

When a developer changes the layout on a template standard value this change is inherited by all items and clones based on this template.

Items with custom layout details are rendered using a combination of the new layout details from the template and the layout customizations made by the user. This difference between the two is the layout delta.

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

ASP.NET supports master pages, which provide some of the features that layouts provide. Master pages (which use the `.aspx` extension) contain content and reference master pages (`.master` files) that control presentation. 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 rather than on a file system. Layout files can 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 ASP.NET content pages provide. Rather than referencing a single master page, a content item can reference different layouts 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 different layouts and renderings for any number of 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 provide a page-based architecture, which can present challenges when developing navigation, reusing content, or implementing other dynamic features. The Sitecore layout engine provides a data-driven architecture that facilitates navigation, reuse, and other dynamic features.

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

3.3.4 Conditional Rendering

You can use conditional rendering to dynamically show or hide renderings and to set rendering properties and parameters. Select conditional rendering rules in the **Personalization** rendering parameter of the rendering in the **Control Properties** dialog. For more information about the **Personalization** rendering parameter, see the section The Personalization Rendering Parameter.

For more information about conditional rendering, see the manual *Rules Engine Cookbook*.

3.4 Presentation Component Definition Items

All presentation components involve files on disk, such as a compiled class in a .NET assembly or an `.ascx`, `.aspx`, `.xslt`, or other code file. Presentation components often depend on CSS, JavaScript, media, and other files on disk, which may or may not have corresponding definition items in Sitecore.

Layouts, sublayouts, XSL renderings, and other types of items consist of a definition item that contains a field named **Path** that specifies the path to file on disk that implements the presentation logic. Fields in web control rendering definition items and method rendering definition items contain information to identify the .NET component for the layout engine to invoke.

Warning
If you move, duplicate, rename, or delete a definition item, Sitecore does not automatically update the **Path** field. If you create, move, duplicate, rename, or delete a file, Sitecore does not automatically create, move, duplicate, rename, or delete the corresponding definition item.

Important
When copying presentation components from one environment to another, include new and updated files and the corresponding definition items.

Chapter 4

Output Caching

This chapter describes features of the layout engine that you can use to cache the output of each presentation component to maximize performance and throughput.

This chapter first describes the various criteria by which you can cache each rendering, and then explains how Sitecore caches rendered output by those criteria.

This chapter contains the following sections:

- Rendered Output Caching Options
- Rendered Output Caching Implementation

4.1 Rendered Output Caching Options

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

The layout engine can cache the output of each rendering. Developers use rendered output caching to improve performance by retrieving output generated previously by that component under the same conditions instead of invoking the component again. While output caching does not eliminate the need for data structure and code optimization, caching can increase performance significantly, especially in high-volume solutions.

Important
Caching is critical to overall solution performance. The most efficient way to increase the throughput and capacity of a Sitecore solution is often 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.

4.2 Rendered Output Caching Implementation

By default, the layout engine executes each presentation component for each HTTP request, without any output caching.

Important
You must select caching criteria each time you use a presentation component.

Note
Without customization, you cannot cache the output of a method rendering or a URL rendering.

The layout engine manages a separate output cache for each managed website. Caching automatically varies by managed website.

Each output cache 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 can generate cache entries with different cache keys representing output generated by a single rendering under different conditions.

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

Important
To cache the output of a web control, override the `GetCachingID()` method to return an identifier for the rendering, such as the GUID of the rendering definition item or the namespace and class name.

Note
Caching can vary by multiple criteria. For example, you can configure caching for a presentation component to vary by both data source and user.

When you set the *VaryBy* properties described in the following sections to *True*, Sitecore 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 cached output if an entry with the corresponding key exists in the cache.

If you do not configure caching for a presentation component, or the cache does not contain a corresponding entry, then the layout engine invokes the component. If the component has caching properties, then the layout engine adds an entry with the corresponding key to the cache.

Important
To minimize both memory usage and the number of times the system must invoke each component, cache the output of each sublayout and rendering by the fewest criteria possible.

Note
By default, publishing clears output caches.

4.2.1 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 presentation component when you statically bind it to a layout or sublayout.
- In the **Caching** section of the **Control Properties** dialog when you bind a presentation component to a placeholder in layout details.

The layout engine uses cache criteria defined in the **Caching** section of the definition item under two conditions:

- 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 rendering definition item to the control (a static reference to the presentation component).
- When layout details do not specify caching criteria for presentation components dynamically bound to placeholders.

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 in the **Caching** section in the **Control Properties** dialog.

4.2.2 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 set to *False*, then the layout engine invokes the component each time it processes a reference to the component. 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 set to *True* and no *VaryBy* properties are set to *True*, then the layout engine invokes the presentation component on its first use by each managed website for each language and writes that output to the cache, then retrieves that cached output for all subsequent uses of that component for that managed site and language. If the *Cacheable* property is *True* and one or more *VaryBy* properties are *True*, then those *VaryBy* properties control whether or not the layout engine invokes the component or retrieves cached output generated previously under the same *VaryBy* conditions.

Set the *Cacheable* attribute:

- To *False* for any renderings for which the layout engine should not cache output.
- To *True* for any 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 set to *False*, then *VaryBy* properties have no effect. The layout engine never caches the output of components for which the *Cacheable* property is set to *False* or undefined.

Clear on Index Update

The *Clear on Index Update* property controls whether or not a control clears its cache when an item that is associated with the control is updated in the index.

This is useful for any control that uses the new Search API to populate its data sources.

For example:

- You have a control that returns all the products that are on special offer from the index.

- The *Clear on Index Update* property for this control is set to *True*.
- The price of each product is stored in the index.
- If someone updates the price of one of the products on special offer, the *Clear on Index Update* property will the trigger the control to clear its cache because something has been updated in the index that is bound to the control.

VaryByData

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

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.

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.

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 the **VaryByLogin** property, 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 instances 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
The **VaryByParm** property causes output caching to vary based on rendering parameter values passed by the developer. The **VaryByQueryString** property 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
The **VaryByUser** treats all anonymous users as a single authenticated user.

Note
To avoid excess memory consumption, avoid using the **VaryByUser** property except in solutions with relatively small numbers of users, or monitor cache utilization closely.

Note
The **VaryByLogin** property causes Sitecore to generate different output depending based on whether or not a user has authenticated, differentiating anonymous users from authenticated users. The **VaryByUser** property causes Sitecore to generate different output for each user.

Chapter 5

Choosing Presentation Technology

This chapter provides guidance for choosing between available presentation component technologies.

This chapter begins by describing general presentation technology considerations, and continues to discuss considerations specific to XSL renderings, sublayouts, and web controls.

This chapter contains the following sections:

- General Presentation Technology Considerations
- Specific Presentation Technology Considerations

5.1 General Presentation Technology Considerations

Consider the following when selecting a technology for each presentation component:

- Layouts represent markup superstructures shared to numerous page views.
- Placeholders represent regions of layouts and sublayouts to which the layout engine dynamically binds renderings according to layout details.
- Existing web forms and web user controls convert most easily to sublayouts.
- Only layouts and sublayouts can contain placeholders.
- Web control renderings can easily wrap or replace existing web controls.
- Method renderings can easily wrap existing .NET methods.
- Web part renderings can easily 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 any presentation component for which the technology choice is not clear based on the considerations above, choose between implementing the component as a sublayout, as a web control rendering, or as an XSL rendering.

Note
Each page can contain presentation components implemented in multiple technologies.

In general, XSL renderings are appropriate for components that generate markup by processing the Sitecore item hierarchy and retrieving field values, while sublayouts and web control renderings are appropriate for presentation components containing significant logic.

5.2 Specific Presentation Technology Considerations

The following sections describe specific advantages and disadvantages of the technologies that you can use to implement each presentation component.

5.2.1 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.

5.2.2 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 syntax, 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, 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 when needed.

XSL uses text files that you can edit in the **Developer Center** or Microsoft **Visual Studio**. When you update an XSL rendering, ASP.NET does not restart as it does when you edit a compiled .NET rendering.

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 Microsoft **Visual Studio**, 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 the Sitecore debugger. XSL renderings do not provide compile-time error detection, only runtime exception management.

XSL renderings execute source code, which you must deploy to all environments including production. Dynamically interpreted XSL does not perform as well as native .NET code.

Whether or not a project uses XSL, Sitecore developers must understand ASP.NET. Because a you can accomplish anything with .NET that you can accomplish with XSL, XSL is an optional technology requiring additional developer skills to implement and support. Using both XSL and .NET can result in similar logic in two languages.

XSL renderings can invoke .NET logic through XSL 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 can outweigh the performance differential, especially for components that generate output that you can cache.

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

5.2.3 Web Control Considerations

The advantages of web controls are similar to those of Sublayouts. For more information about the advantages of sublayouts, see the section Sublayout 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.

Unlike sublayouts, web controls cannot contain placeholders, but you can add controls programmatically. More importantly, sublayouts do not separate design from presentation using the ASP.NET code files. Web controls are appropriate for components that generate markup dynamically with little or no predefined markup.