Sitecore CMS 6

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 Sitecore's layout engine as well as detailed explanations of the Sitecore presentation components for CMS architects, developers, and administrators.

This document assists in the identification and definition of the presentation components required to implement a Sitecore solution. Understanding how the layout engine enhances the ASP.NET page assembly process aids in defining requirements data structures and reusable presentation components, as well as diagnosing problems with a Web site. Effective implementation of the layout engine minimizes development, maintenance and administration costs.

This document contains the following chapters:

Chapter 1 — Introduction
Introduces the topics discussed in this document.

Chapter 2 — Presentation Components
Describes the presentation components used by the Sitecore layout engine to service HTTP requests.

Chapter 3 — Request Processing
Describes the Sitecore layout engine enhancements to the ASP.NET page assembly process.

Chapter 4 — Developer Center and Microsoft Visual Studio
Describes how developers use Sitecore Developer Center and Microsoft Visual Studio to maintain presentation components.

Chapter 5 — Output Caching
Describes how developers use presentation component caching options to increase performance.

Chapter 6 — Choosing Presentation Technology
Guidelines for choosing the appropriate rendering technology for a specific presentation component.

Chapter 2

Presentation Components

This chapter contains a short overview of the layout engine and detailed information about the presentation components it uses.

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

Sitecore's layout engine extends the ASP.NET Web application server to separate content from presentation until Sitecore generates a response to a page request. The layout engine dynamically merges data with code according to layout details defined in the requested content item, automatically accounting for the user's context such as language, device, and access rights.

The Sitecore layout engine uses ASP.NET. 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 or the standard values item associated with the item's data template control the presentation components the layout engine uses to service HTTP requests from different types of devices. Each presentation component can access the entire CMS database, 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, and ASP.NET page and control events. 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 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 items 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 superstructures shared to any number of responses to HTTP requests, such as the outer structure including a header and footer shared to all or most pages on a Web site. Developers use layouts:

- To define the outermost markup shared to the greatest number of page views.
- To define different outer dimensions for different pages, such as a specific form factor for a site's home page that differs from all other pages.
- To apply different presentations when different types of devices request a content item, such as formatting content differently for browsers and PDAs.

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. Layouts definition items under `/Sitecore/Layout/Layouts` use the `/Sitecore/Templates/System/Layout/Layout` data template. The *Path* property 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 controls. Some controls are static literals, and others are dynamic server controls. The layout engine passes literals directly through to the response stream as markup elements. Server controls respond to page events and generate output dynamically. Server controls used with Sitecore include:

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

Developers bind some server controls to layouts statically, at design time, by adding controls to layouts and sublayouts. Developers bind other server controls to placeholders in a layout dynamically, at runtime, using layout details.

2.2.2 Layout Usage

Each HTTP request activates at most a single layout. Sitecore and ASP.NET process some requests using other ASP.NET handlers that do not involve Sitecore layouts. Each logical Web site usually involves at least one layout per device. Layouts are generally the most reusable presentation component. Some Sitecore solutions use a single layout for all page views.

Developers bind presentation components to the layout statically if they execute on every invocation of a layout.

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.

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

Note
Global logic generally belongs in pipelines or `global.asax`. Application logic belongs in application components such as sublayouts. Layouts generally do not use .NET code-behind and code-beside facilities.

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 a sublayout. Developers use sublayouts:

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

2.3.1 Sublayout Implementation

ASP.NET uses `.ascx` files, known as Web user controls, to service some portions of HTTP requests. Sitecore sublayouts are ASP.NET Web user controls registered with the content management system, including both a sublayout definition item and an `.ascx` file. Sublayout definition items under `/Sitecore/Layout/Sublayouts` use the `/Sitecore/Templates/System/Layout/Sublayout` data template. The *Path* property in each layout definition item references the path to an `.ascx` file relative to the document root of the Web site.

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.

Each HTTP request invokes zero or more sublayouts to service portions of the request. The layout engine dynamically binds presentation components to placeholders in the layout and nested placeholders in sublayouts as specified in layout details.

2.3.2 Sublayout Usage

While developers can statically bind sublayouts to layouts and sublayouts, developers more frequently bind sublayouts to placeholders in layouts and sublayouts. Sublayouts may contain placeholders supporting nesting 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 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 component, such as request logging.

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.

2.4.2 Rendering Usage

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

Sublayouts can function as renderings 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. Developers can statically bind Web controls that inherit from the standard 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 that make the browser request a URL, URL renderings request the URL on the server. 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>`.

Web Part Renderings

The optional Web parts framework allows developers to use of Web parts as renderings.[1]

2.4.4 Rendering Data Source

Each rendering can accept a source item specifying a location in a Sitecore database from which the rendering should begin processing. Developers pass a data source to a rendering:

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

The context item, which corresponds to the requested URL, is the default data source for all renderings for which the developer does not specify a data source.

[1] For more information about using Web parts with Sitecore, see http://sdn5.sitecore.net/Resources/Free%20Modules/Web%20Part%20Framework.aspx.

2.4.5 Rendering Parameters

Each rendering can accept any number of parameters. Developers pass parameters to renderings:

- To avoid hard-coding content or configuration in a rendering.
- To reuse renderings with different configurations.

2.4.6 Field Renderer Web Control

Sitecore provides the field renderer Web control to output a single field value, automatically providing inline editing controls in the Page Editor. Developers use the field renderer Web control to retrieve and format a single field value.

The FieldRenderer Web control 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 of the field.
EnclosingTag	Markup element to wrap field value (for example, div).
FieldName	Name of field to process.

The FieldRenderer Web control also supports the parameters defined by the Sitecore Web control base class. This base class defines the `DataSource` parameter, which controls the item from which Sitecore retrieves the field value. If the developer does not specify the `DataSource` parameter, the layout engine retrieves the field value from the context item.

The `Sitecore.Web.UI.WebControls.FieldRenderer` class in the `Sitecore.Kernel` assembly provides the implementation of the field renderer 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 according to layout details.[2] 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 named placeholders with the various sublayouts and renderings associated with that key in the layout details for the requested item.

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.

The `LayoutPageEvent` setting in `web.config` controls which ASP.NET page event causes the layout engine to apply layout details. Developers can choose to bind sublayouts and renderings to placeholders during the `PreInit` event, the `Init` event, or the `Load` event.

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.

2.5.2 Placeholder Keys

Each placeholder has a textual key. Layout details reference a placeholder key for each presentation component. These references instruct the layout engine which presentation components to bind dynamically to the placeholder when generating page views, and in what order to bind those components to that placeholder.

Placeholder keys must be unique within all of the presentation components referenced in the layout details for 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. Placeholders in common regions defined in multiple components never used together on 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

[2] 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.

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 control the sublayouts and renderings users can bind to a placeholder. For more information about placeholder settings, see the Client Configuration Reference manual.

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. Statically bind presentation components whenever possible.

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 of functionality provided by the layout engine beyond those provided by the ASP.NET Web application server.

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 three primary ways:

- A .NET HTTP module maps URLs to content items in the database instead of files on disk.
- The HTTP module defines a context object indicating the user, language, requested content item, and other contextual information.
- The HTTP module applies the layout details specified in the requested content item to generate output.

Web applications respond to HTTP requests from Web browsers, PDAs, RSS readers and other devices. The Sitecore layout engine uses properties of each HTTP request, such as the domain, the path, query string parameters, and HTTP headers such as browser user agent and cookies, to determine how to assemble the response. The layout engine applies the presentation components referenced in the layout details of the requested content item. These ASP.NET and XSL presentation components assemble the response by accessing the CMS database and any other resources available to ASP.NET and XSL.

3.1.1 The Context Item

The layout engine identifies a content item in the database based on the path in the requested URL. The requested item becomes the context item for the lifecycle of the request. The context item is the default item in the Sitecore database for most operations associated with the page request, such as determining layout details to apply. The context item is the default data source for sublayouts and renderings for which the developer does not specify a data source.

For example, under the default configuration, if the browser requests `/hr/jobs.aspx`, the layout engine sets the context item to the content item `/Sitecore/Content/Home/hr/jobs`.

3.2 Devices

Devices represent the different types of clients connected to the Internet that place HTTP requests against the Web server. Using devices and layout details, the layout engine applies different presentation components to a content item based on properties of the request. Developers use devices to process requests for content items using different presentation components for different browsers, printers, PDAs, RSS readers, and other types of devices with various form factors and other markup requirements.

3.2.1 Device Implementation

The layout engine determines a device for each HTTP request, known as the context device. If the properties of a request do not identify a specific device, the context device is the Default device, which typically represents a Web browser.

The layout engine determines the requesting device based on:

- Query string parameters.
- Specific user agents.
- The requested hostname.
- .NET logic.

Fallback Device

Developers may associate each device with a fallback device. The layout engine applies the layout details for the fallback device if the context item does not contain layout details for the context device determined from request properties.

Note
If defined, layout details for the fallback device in the context item override layout details for the context device in the standard values item associated with the context item's data template. 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 those layout details without checking for layout details for the context device in the standard values item for the context item's template.

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 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.

Developers may create any number of additional devices to represent additional types of clients, or other criteria requiring the layout engine to format content differently. Additional devices may include but are not limited to the following:

- RSS readers.
- Mobile devices.

- Flash, such as to consume XML from the CMS.
- Multiple logical Web sites.

Note
Developers must define request properties or logic to trigger any custom devices.

3.3 Layout Details

Layout details contain references to reusable presentation components for the layout engine to invoke when servicing requests for content items from different types of devices. Developers use layout details:

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

3.3.1 Layout Details Implementation

ASP.NET Web applications map URLs in HTTP requests to files on disk. For example, a request for `/hr/jobs.aspx` invokes processing of the file `jobs.aspx` in the `/hr` directory under the Web site's document root.

The Sitecore layout engine maps URLs to content items in a database. Layout details in each content item reference layout, sublayout and rendering definition items to process when different devices request the item.

The standard template inherited by all content item 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.

Different items dynamically populate the same placeholders in common layouts and sublayouts with different components at runtime. Each component generates output dynamically based on data in a Sitecore database, using Sitecore APIs and any other resources or APIs available to ASP.NET or XSL.

Layout details separate data from presentation, 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 item associated with each data template used to create content items.

3.3.2 Layout Details vs. ASP.NET Master Pages

ASP.NET supports content pages (`.aspx` files) that reference master pages (`.master` files) controlling presentation of the content.[3] ASP.NET master pages contain controls applied 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 content 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 the ASP.NET master page infrastructure. Instead of referencing a single master page, a content item may reference

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

different layouts, sublayouts, and renderings to invoke when the different types of devices request the item.

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.

With just ASP.NET master pages, formatting content differently for different devices would require custom logic or multiple content pages, often resulting in content and markup duplication. Users may translate Sitecore content items into any number of languages, all using the same presentation components. With just master pages, translation would require custom logic or multiple content pages.

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 some specific release management considerations for 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

Sitecore's browser-based Developer Center allows developers to edit layouts, sublayouts, and XSL renderings. Developers may also edit presentation components in offline file editors such as Visual Studio.

Developer Center has some advantages over Visual Studio:

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

Creating a layout, sublayout, or XSL rendering in Developer Center invokes a wizard that copies a Sitecore boilerplate for the file type to the new location and creates a corresponding definition item, involving less steps than creating the item in Visual Studio and then manually creating the corresponding definition item in Sitecore.

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

- A single development environment for all types of resources including C#, VB.NET, and other languages supported by .NET.
- Intellisense, syntax completion, automatic code indentation, error underlining, and other integrated development environment features.
- The Visual Studio debugger for .NET code.
- Integration with source code management tools.

Note
The ASP.NET 2.0 Web Application project model available in Visual Studio 2005 Service Pack 1 and Visual Studio 2008 is convenient for most Sitecore solutions.[4]

[4] For instructions to create a Visual Studio Web Application project for use with a Sitecore solution, see http://sdn5.sitecore.net/Articles/API/Creating%20VS2005%20Project.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 drags a sublayout or rendering onto a layout or sublayout in Developer Center, the system copies caching properties from the definition item to the new static reference.

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

Cache settings must be explicitly defined wherever a developer statically binds a presentation component to a layout or sublayout. When a component is dynamically bound to a placeholder, cache settings explicitly defined on the caching tab override cache settings defined in the 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 the browser or any text editor, 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.