Sitecore CMS 7.0

Presentation Component API Cookbook

A Conceptual Overview for CMS Developers

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Table of Contents

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Chapter 1

Introduction

This document describes Application Programming Interfaces (APIs) that you can use in presentation components. You should read this document before developing Sitecore layouts, sublayouts, or web controls, and implement .NET extensions for XSL renderings.

This document begins with an overview of the Sitecore context, which contains information about the current HTTP request and controls Sitecore processing. The next two chapters demonstrate APIs that you can use to access items and field values. The following chapter presents APIs that you can use to work with multiple languages of each item. The remaining chapter provides techniques for error and exception management.

This document contains the following chapters:

- Chapter 1 — Introduction
- Chapter 2 — The Sitecore Context
- Chapter 3 — Accessing Items
- Chapter 4 — Accessing Fields
- Chapter 5 — Working with Languages
- Chapter 6 — Error Management

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Chapter 2

The Sitecore Context

This chapter provides an overview of the Sitecore context. Beginning with an overview of the Sitecore context, this chapter continues to describe the context item, site, database, language, device, domain, user, raw URL, and page mode.

This chapter contains the following sections:

- Overview of the Sitecore Context
- The Context Item
- The Context Site
- The Context Database
- The Context Language
- The Context Device
- The Context Domain
- The Context User
- The Context Raw URL
- The Context Page Mode

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.1 Overview of the Sitecore Context

The Sitecore context contains information about the current HTTP request, and control Sitecore features including the layout engine. This section describes properties of the `Sitecore.Context` class that define the Sitecore context.

The layout engine invokes the `httpRequestBegin` pipeline defined in `web.config` to define the Sitecore context for each HTTP request. You can use APIs to set Sitecore context properties from any code that runs in a Sitecore context, such as any .NET presentation component. You can add, remove, and replace `httpRequestBegin` pipeline processors to customize how Sitecore defines the context.

Note
Some code examples in this document assume the existence of a `System.Web.UI.HtmlTextWriter` object named `output`.

For more information about presentation components, see the manuals *Presentation Component Reference* and *Presentation Component Cookbook*.

For information about using ASP.NET login controls and Sitecore APIs to authenticate website users and manage user profiles, see the manual *Security API Cookbook*.

For more information about the APIs described in this document, see the *Content API Cookbook k*.

For more information about additional Sitecore APIs, see the manual *Sitecore API Reference*.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.2 The Context Item

The context item is the Sitecore item activated by the current HTTP request. The context item typically corresponds to the path in the URL of the current HTTP request. The item resolver processor (`Sitecore.Pipelines.HttpRequest.ItemResolver`) in the `httpRequestBegin` pipeline sets the context item.

The context item typically corresponds to the path in the current URL. For example, under the default configuration, Sitecore sets the context item to `/Sitecore/Content/Home/Section/Page` when it processes a URL that contains the path `/section/page.aspx`.

Note
The context item does not always correspond directly to the path in the current URL. For example, when an HTTP request activates an alias, the URL corresponds to the alias definition item, but the context item corresponds to the item referenced by that alias definition item.

For more information about aliases, see the manual *Content Reference*.

You can use the `Sitecore.Data.Items.Item` class and the `Sitecore.Context.Item` property to access the context item. For example:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
```

Many presentation components depend on the context item. For example, a content body rendering can retrieve field values from the context item, a breadcrumb rendering can output links to ancestors of the context item, and a navigation rendering can style links to the context item and its ancestors to indicate the location of the context item in the information architecture.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.3 The Context Site

The context site is the managed website activated by the current HTTP request. Each `/configuration/sitecore/sites/site` element in `web.config` defines the properties of a managed website. Comments in `web.config` document the purpose of each attribute. The site resolver (`Sitecore.Pipelines.HttpRequest.SiteResolver`) in the `httpRequestBegin` pipeline sets the context site to the first site definition that matches properties of the HTTP request.

You can use the `Sitecore.Sites.SiteContext` class and the `Sitecore.Context.Site` property to access the context site. For example:

```
        Sitecore.Sites.SiteContext contextSite = Sitecore.Context.Site;
```

Presentation components can use the properties of the context for a variety of purposes.

2.3.1 The Home Item of the Context Site

The home item for a managed website corresponds to the home page of that site.

You can use the `Sitecore.Context.Site.StartPath` property to access the home item of the context site. For example, to determine the home item of the context site:

```
        Sitecore.Sites.SiteContext contextSite = Sitecore.Context.Site;
        Sitecore.Data.Items.Item home =
Sitecore.Context.Database.GetItem(contextSite.StartPath);
```

Some presentation components depend on the home item. For example, in a logo rendering, you can retrieve an image from the home item and include a link to the home item, and a navigation rendering can output links to each of the children of the home item.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.4 The Context Database

The context database is the Sitecore database associated with the current HTTP request. The database resolver (`Sitecore.Pipelines.HttpRequest.DatabaseResolver`) in the `httpRequestBegin` pipeline sets the context database. Each managed website can specify the default context database for that site; the default context database is that associated with the context site. Sitecore user interfaces and URL query string parameters can cause the context database to differ from the default context database specified by the context site.

You can use the `Sitecore.Data.Database` class and the `Sitecore.Context.Database` property to access the context database. For example:

```
Sitecore.Data.Database contextDatabase = Sitecore.Context.Database;
```

Important
In order to function properly in all environments, presentation components always use the context database instead of referencing a database by name.

Important
Presentation components do not update Sitecore databases.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.5 The Context Language

The context language is the language associated with the current HTTP request. The language resolver (`Sitecore.Pipelines.HttpRequest.LanguageResolver`) in the `httpRequestBegin` pipeline sets the context language. Unless otherwise specified, APIs retrieve field values from the current version of each item in the context language.

You can use the `Sitecore.Globalization.Language` class and the `Sitecore.Context.Language` property to access the context language. For example:

```
Sitecore.Globalization.Language contextLanguage = Sitecore.Context.Language;
```

The getter for the context language property returns the first defined value from the following list:

- The value of the property.
- The value of the Sitecore language cookie associated with the context site.
- The value of the `language` attribute in the context site definition.
- The `value` attribute of the `/configuration/sitecore/settings/setting` element in `web.config` with name `DefaultLanguage`.

The language resolver sets the context language if the `sc_lang` query string parameter specifies a valid language, or if the `languageEmbedding` attribute of the link manager configuration is a value other than `never` and the first step in the path in the URL specifies a language. For examples of custom pipeline processors that set the context language based on additional criteria, see the section *Example: Determine Context Language from the User Profile* and the section *Example: Determine Language from Web Client Preferences*.

For more information about link management configuration, see the manual *Content API Cookbook*.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.6 The Context Device

The context device is the device associated with the current HTTP request. The device resolver (`Sitecore.Pipelines.HttpRequest.DeviceResolver`) in the `httpRequestBegin` pipeline sets the context item. The layout engine processes the request using the presentation components specified by the layout details for the context device in the context item.

You can use the `Sitecore.Data.Items.DeviceItem` class and the `Sitecore.Context.Device` property to access the context device. For example:

```
Sitecore.Data.Items.DeviceItem contextDevice = Sitecore.Context.Device;
```

If the context device property is undefined, the device resolver sets the context device to the first value from the following list that specifies a valid device:

- The `sc_device` query string parameter.
- The `device` attribute of the context site.
- The first device in the context database that matches the URL query string parameters or web client user-agent associated with the current HTTP request.
- The `defaultDevice` attribute of the context site.
- The first device with the Default checkbox field selected in the Data section.

Note
Use the `device` attribute in a managed website definition to cause that device to be the context device for all requests associated with that website that do not include the `sc_device` query string parameter. Use the `defaultDevice` attribute to specify different default devices for different managed websites.

2.6.1 Example: Set the Context Device

You can implement a pipeline processor based on the following example that sets the context device to the device named Feed if the web client user-agent string matches the user-agent of one of several common RSS readers:

```
namespace Sitecore.Sharedsource.Pipelines.HttpRequest
{
  public class FeedDeviceResolver
  {
    public void Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
    {
      System.Web.HttpContext httpContext = System.Web.HttpContext.Current;

      if (httpContext == null || httpContext.Request == null
        || httpContext.Request.UserAgent == null || Sitecore.Context.Database == null)
      {
        return;
      }

      string agent = httpContext.Request.UserAgent;

      if (agent.Contains("feedfetcher")
          || agent.Contains("newsgatoronline")
          || agent.Contains("bloglines")
          || agent.Contains("yahoofeedseeker"))
      {
        Sitecore.Data.Items.DeviceItem device =
          Sitecore.Context.Database.Resources.Devices["Feed"];

        if (device != null)
        {
          Sitecore.Context.Device = device;
        }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
            }
          }
        }
      }
```

Insert the processor before the device resolver within the `/configuration/sitecore/pipelines/httpRequestBegin` element in `web.config`

```
      <httpRequestBegin>
      ...
        <processor
          type="Sitecore.Sharedsource.Pipelines.HttpRequest.FeedDeviceResolver, Assembly" />
        <processor type="Sitecore.Pipelines.HttpRequest.DeviceResolver, Sitecore.Kernel" />
      ...
```

For more information about Sitecore RSS features, see the manual *Presentation Component Cookbook*.

For information about RSS APIs, see the manual *Content API Cookbook*.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.7 The Context Domain

The context domain is the Sitecore security domain associated with the context site. Sitecore determines the context domain dynamically without using a pipeline processor. Each managed website can specify the default context domain for that site; the context site defines the default context domain. Sitecore user interfaces and URL parameters can cause the context domain to differ from that associated with the context site. Sitecore user interfaces and URL query string parameters can cause the context domain to differ from the default context domain specified by the context site.

For more information about security APIs, see the manual *Security API Cookbook*.

You can use the `Sitecore.Security.Domains.Domain` class and the the `Sitecore.Context.Domain` property to access the context domain. For example:

```
Sitecore.Security.Domains.Domain contextDomain = Sitecore.Context.Domain;
```

Presentation components that use security APIs, such as registration and login forms, often access the context domain.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.8 The Context User

The context user is the security user associated with the current HTTP request. The user resolver (`Sitecore.Pipelines.HttpRequest.UserResolver`) in the `httpRequestBegin` pipeline sets the context user. If the user is not authenticated, then the context user is the anonymous user in the context domain.

You can use the `Sitecore.Security.Accounts.User` class and the `Sitecore.Context.User` property to access the context user. For example:

```
Sitecore.Security.Accounts.User contextUser = Sitecore.Context.User;
```

You can use the `Sitecore.Context.IsLoggedIn()` method to determine whether the user is authenticated. For example:

```
if (Sitecore.Context.IsLoggedIn)
{
  //TODO: handle case that user is authenticated
}
else
{
  //TODO: handle case that user is not authenticated
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.9 The Context Raw URL

The context raw URL provides access to the path and query string components of the current HTTP request, excluding the protocol and hostname. Sitecore determines the raw URL without using an `httpRequestBegin` pipeline processor. For example, given the URL `http://localhost/path/item.aspx?q=v`, the context raw URL contains `/path/item.aspx?q=v`.

You can use the `Sitecore.Context.RawUrl` property to access the context raw URL. For example:

```
string rawUrl = Sitecore.Context.RawUrl;
```

You can include the raw URL when logging various conditions. For example:

```
string message = String.Format("{0} : {1}", GetType(), Sitecore.Context.RawUrl);
Sitecore.Diagnostics.Log.Info(message, this);
```

You can use the `Sitecore.Web.WebUtil.GetLocalPath()` method with the `Sitecore.Context.RawUrl` property to access the path in the raw URL. For example:

```
string rawUrl = Sitecore.Context.RawUrl;
string path = Sitecore.Web.WebUtil.GetLocalPath(rawUrl);
```

You can use the `Sitecore.Web.WebUtil.ParseQueryString()` method with the `Sitecore.Context.RawUrl` property to parse the query string into a collection. For example, to retrieve the value of the URL query string parameter named `key`:

```
string rawUrl = Sitecore.Context.RawUrl;
Sitecore.Collections.SafeDictionary<string> queryString =
  Sitecore.Web.WebUtil.ParseQueryString(rawUrl);
string value = queryString["key"];

if (value == null)
{
  output.Write("//TODO: handle case that query string does not include parameter");
}
else if (String.IsNullOrEmpty(value))
{
  output.Write("//TODO: handle case that parameter is empty");
}
else
{
  output.Write("//TODO: process value");
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2.10 The Context Page Mode

The page mode indicates active Sitecore features for the current HTTP request. Sitecore determines the page mode dynamically without using a pipeline processor. Presentation components can generate different output in different page modes, such as on the published site, in the Page Editor, and in the Debugger.

For more information about the page mode, see the manual *Client Configuration Reference*.

You can use the `Sitecore.Context.PageMode` property to determine the page mode. For example, to determine if the web client is in the Page Editor:

```
if (Sitecore.Context.PageMode.IsPageEditor)
{
  //TODO: handle case that user is in the Page Editor
}
```

2.10.1 Example: Expose Metadata Fields

In some cases, metadata fields provide no visual page element on the published website. You can use the FieldRenderer web control as described in the section The FieldRenderer Web Control to enable inline editing controls for metadata fields only when the user is editing in the Page Editor. For example, to edit the Title field in the context item, add a FieldRenderer web control such as the following to a layout or sublayout file:

```
<sc:FieldRenderer runat="server" id="fldTitle" FieldName="title" visible="false" />
```

You can use the `Page_load()` method in the layout or sublayout code file to enable the FieldRenderer web control if the user is editing in the Page Editor:

```
protected void Page Load(object sender, EventArgs e)
{
  if (Sitecore.Context.PageMode.IsPageEditorEditing)
  {
    fldTitle.Visible = true;
  }
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Chapter 3

Accessing Items

This chapter provides information about APIs that you can use to access items. Beginning with an overview of accessing items, this chapter continues to describe populating ASP.NET controls with data from items and accessing the descendants and ancestors of an item before concluding with information about sorting items.

This chapter contains the following sections:

- Overview of Accessing Items
- How to Populate ASP.NET Controls with Item Values
- How to Access the Descendants of an Item
- How to Access the Ancestors of an Item
- Sorting Items

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

3.1 Overview of Accessing Items

This section provides an overview of APIs that you can use to access items.

For more information about using APIs to access items, see the manual *Content API Cookbook*.

3.1.1 How to Access the Friendly URL of a Content Item

You can use the `Sitecore.Links.LinkManager.GetItemUrl()` method to access the friendly URL of a content item. For example, to access the friendly URL of the context item:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
string url = Sitecore.Links.LinkManager.GetItemUrl(contextItem);
```

For more information about dynamic link management, see the manual *Content API Cookbook*.

3.1.2 How to Access the Friendly URL of a Media Item

You can use the `Sitecore.Resources.Media.MediaManager.GetMediaUrl()` method to access the friendly URL of a media item. For example, to access the friendly URL of the media item referenced by the Image field named ImageField in the context database:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
Sitecore.Data.Fields.ImageField image = contextItem.Fields["imagefield"];

if (image.MediaItem != null)
{
  string url = Sitecore.StringUtil.EnsurePrefix('/',
    Sitecore.Resources.Media.MediaManager.GetMediaUrl(image.MediaItem));
}
```

For more information about media URLs management, see the manual *Content API Cookbook*.

3.1.3 How to Determine if an Item Contains Layout Details for a Device

You can use the `Sitecore.Data.Items.Item.Visualization.GetLayout()` method to determine whether an item contains layout details for a device. The `Sitecore.Data.Items.Item.Visualization.GetLayout()` method returns Null if the item does not contain layout details for the specified device. For example, to generate a list of links to each of the children of the context item that contain layout details for the context device:

```
System.Text.StringBuilder markupBuilder = new System.Text.StringBuilder();
foreach (Sitecore.Data.Items.Item child in Sitecore.Context.Item.Children)
{
  Sitecore.Data.Items.LayoutItem layout =
    child.Visualization.GetLayout(Sitecore.Context.Device);
  if (layout != null)
  {
    string url = Sitecore.Links.LinkManager.GetItemUrl(child);
    markupBuilder.AppendFormat(@"<li><a href=""{0}"">{1}</a></li>", url, child.Name);
  }
}
if (markupBuilder.Length > 0)
{
  markupBuilder.Insert(0, "<ul>");
  markupBuilder.Append("</ul>");
  string markup = markupBuilder.ToString();
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

3.2 How to Populate ASP.NET Controls with Item Values

You can populate an ASP.NET control such as a drop-down list with item data using the APIs described in this section.

3.2.1 How to Bind the Children of an Item to an ASP.NET Control

You can bind the children of an item to an ASP.NET control using the `Sitecore.Data.Items.Item.Children` property. For example, to populate an ASP.NET drop-down list with options containing the names of the children of an item, using the IDs of those items as the values of those options, add a drop-down list control such as the following to a layout or sublayout file:

```
<asp:dropdownlist runat="server" id="lstDropDown" />
```

Implement data binding for the drop-down list in the `Page_Load()` method in the layout or sublayout code file:

```
protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)
  {
    Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
    lstDropDown.DataSource = contextItem.Children;
    lstDropDown.DataTextField = "name";
    lstDropDown.DataValueField = "id";
    lstDropDown.DataBind();
  }
}
```

How to Bind an List of Items to a Control

You can bind a list of `Sitecore.Data.Items.Item` objects to an ASP.NET control. For example, to populate an ASP.NET drop-down list with options containing the names of all descendants of the `/Sitecore/Content` item in the context database that are based on the `Common/Folder` data template, using the IDs of those items as the values of those options, add a drop-down list control such as the following to a layout or sublayout file:

```
<asp:dropdownlist runat="server" id="lstDropDown" />
```

Implement data binding for the drop-down list in the `Page_Load()` method in the layout or sublayout code file:

```
protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)
  {
    string query = String.Format(
      "/sitecore/content//*[@@templateid='{0}']", Sitecore.TemplateIDs.Folder);
    Sitecore.Data.Items.Item[] queried = Sitecore.Context.Database.SelectItems(query);

    if (queried != null)
    {
      lstDropDown.DataSource = queried;
      lstDropDown.DataTextField = "name";
      lstDropDown.DataValueField = "id";
      lstDropDown.DataBind();
    }
  }
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Note
This example uses the descendant axis ("`//`"), which can become inefficient as the number of descendants grows. For solutions involving a large number of items, consider alternative information architectures to prevent use of the descendant axis.

3.2.2 How to Use Field Values when Binding Items to a Control

You can use field values when binding a list of items to an ASP.NET control. For example, to populate an ASP.NET drop-down list with options containing the values of the Title field in the children of the context item, using the IDs of those items as the values of those options, add a drop-down list control such as the following to a layout or sublayout file:

```
        <asp:dropdownlist runat="server" id="lstDropDown" />
```

You can populate the drop-down list in the `Page_Load()` method in the layout or sublayout code file:

```
        protected void Page_Load(object sender, EventArgs e)
        {
          if (!IsPostBack)
          {
            Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;

            foreach (Sitecore.Data.Items.Item child in contextItem.Children)
            {
              System.Web.UI.WebControls.ListItem option =
                new System.Web.UI.WebControls.ListItem(child.Name, child.ID.ToString());
              lstDropDown.Items.Add(option);
            }
          }
        }
```

3.2.3 How to Populate an ASP.NET Repeater with Item Data

You can populate an ASP.NET repeater with data from a list of `Sitecore.Data.Items.Item` objects. For example, to generate an HTML table listing the `Sitecore.Data.Items.Item.Name` property and the value of the Title field in each of the children of the context item, add a repeater such as the following to a layout or sublayout file:

```
        <table border="1">
          <tr>
            <th>Item</th>
            <th>Title</th>
          </tr>
          <asp:Repeater ID="repeater" runat="server">
            <ItemTemplate>
              <tr>
                <td>
                  <asp:Label runat="server" ID="Name"
                    Text='<%#DataBinder.Eval(Container.DataItem,"Name")%>'></asp:Label>
                </td>
                <td>
                  <asp:Label runat="server" ID="Title"

Text='<%#DataBinder.Eval(Container.DataItem,@"[""title""]")%>'></asp:Label>
                </td>
              </tr>
            </ItemTemplate>
          </asp:Repeater>
        </table>
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

You can populate the repeater in the `Page_Load()` method in the layout or sublayout code file:

```
protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)
  {
    repeater.DataSource = Sitecore.Context.Item.Children;
    repeater.DataBind();
  }
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

3.3 How to Access the Descendants of an Item

You can use a recursive method to access the descendants of an item as demonstrated in this section.

3.3.1 Example: Site Map Web Control

You can implement a web control based on the following example that accesses the descendants of the home item of the context site to generate a simple data-driven site map:

```
using System;

namespace Sitecore.Sharedsource.Web.UI.WebControls
{
  public class SiteMap : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(System.Web.UI.HtmlTextWriter output)
    {
      Sitecore.Data.Items.Item home =
        Sitecore.Context.Database.GetItem(Sitecore.Context.Site.StartPath);
      this.SiteMapStep(home, output, 1);
    }

    private void SiteMapStep(
      Sitecore.Data.Items.Item item, System.Web.UI.HtmlTextWriter output, int level)
    {
      if (this.HasVisibleChildren(item))
      {
        output.Write("<ul>");

        foreach (Sitecore.Data.Items.Item child in item.Children)
        {
          if (IsVisible(child))
          {
            string markup = String.Format(
              @"<li class=""{0}""><a href=""{1}"">{2}</a>",
              "sitemap level " + level,
              Sitecore.Links.LinkManager.GetItemUrl(child),
              child.Name);
            output.Write(markup);
            this.SiteMapStep(child, output, level + 1);
            output.Write("</li>");
          }
        }

        output.Write("</ul>");
      }
    }

    private bool IsVisible(Sitecore.Data.Items.Item item)
    {
      return item.Versions.Count > 0
        && item.Visualization.GetLayout(Sitecore.Context.Device) != null;
    }

    private bool HasVisibleChildren(Sitecore.Data.Items.Item item)
    {
      foreach (Sitecore.Data.Items.Item child in item.Children)
      {
        if (this.IsVisible(child))
        {
          return true;
        }
      }

      return false;
    }
  }
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

The site map web control passes the home item of the context site to the recursive method `SiteMapStep()`. If that item has children that have layout details and version data for the context language, then the `SiteMapStep()` method links to each of those children, and if that child has children visible by the same criteria, passes that child recursively to `SiteMapStep()`.

For instructions to implement a web control, see the manual *Presentation Component Cookbook*.

Note
This example provides a visual site map for people browsing the site, not a search engine site map used to describe the structure of the website to search engines.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

3.4 How to Access the Ancestors of an Item

You can use the `Sitecore.Data.Items.Item.Axes.GetAncestors()` method and the `Sitecore.Data.Items.Axes.IsDescendantOf()` methods to work with the ancestors of an item as described in this section.

Note
You can also access the ancestors of an item recursively using the `Sitecore.Data.Items.Item.Parent` property.

3.4.1 Example: Breadcrumb Web Control

A breadcrumb indicates the location of the current page in the information architecture of the website, often with links to each of the levels above the item in that information architecture. You can implement a web control based on the following example that accesses the ancestors of an item to generate a simple breadcrumb:

```
using System;

namespace Sitecore.Sharedsource.Web.UI.WebControls
{
  public class Breadcrumb : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(System.Web.UI.HtmlTextWriter output)
    {
      Sitecore.Data.Items.Item home =
        Sitecore.Context.Database.GetItem(Sitecore.Context.Site.StartPath);

      foreach (Sitecore.Data.Items.Item item in
        Sitecore.Context.Item.Axes.GetAncestors())
      {
        if (item.ID == home.ID || item.Axes.IsDescendantOf(home))
        {
          if (item.Visualization.GetLayout(Sitecore.Context.Device) != null
            && item.Versions.Count > 0)
          {
            output.Write(String.Format(@"<a href=""{0}"">{1}</a>",
              Sitecore.Links.LinkManager.GetItemUrl(item), item.Name));
          }
          else
          {
            output.Write(item.Name);
          }

          output.Write(" &gt; ");
        }
      }

      output.Write(Sitecore.Context.Item.Name);
    }
  }
}
```

The breadcrumb web control begins with the home item of the context site. The breadcrumb includes each ancestor of the context item that is the home item or a descendant of the home item, in document order (from the top of the hierarchy down). If the item has layout details and version data for the context language, the breadcrumb links to that item, otherwise the breadcrumb displays the name of the item. The breadcrumb includes a spacing element between each step, and finally the name of the context item.

For instructions to implement a web control, see the manual *Presentation Component Cookbook*.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

3.4.2 Example: Styled Navigation Web Control

You can implement a web control based on the following example that styles navigation links differently if they link to the context item or one of its ancestors:

```
using System;

namespace Sitecore.Sharedsource.Web.UI.WebControls
{
  public class TopNav : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(System.Web.UI.HtmlTextWriter output)
    {
      Sitecore.Data.Items.Item home =
        Sitecore.Context.Database.GetItem(Sitecore.Context.Site.StartPath);
      output.Write("<ul>");

      foreach (Sitecore.Data.Items.Item child in home.Children)
      {
        if (child.Visualization.GetLayout(Sitecore.Context.Device) != null
          && child.Versions.Count > 0)
        {
          string cssClass = "topnav default";

          if (Sitecore.Context.Item.ID == child.ID)
          {
            cssClass = "topnav_selected";
          }
          else if (Sitecore.Context.Item.Axes.IsDescendantOf(child))
          {
            cssClass = "topnav_current";
          }

          string markup = String.Format(
            @"<li class=""{0}""><a href=""{1}"">{2}</a></li>",
            cssClass,
            Sitecore.Links.LinkManager.GetItemUrl(child),
            child.Name);
          output.Write(markup);
        }
      }

      output.Write("</ul>");
    }
  }
}
```

The navigation web control generates a list of links to each of the children of that home item of the context site that have layout details and version data for the context language. The control applies different CSS classes if the user has requested the item or a descendant of that item.

For instructions to implement a web control, see the manual *Presentation Component Cookbook*.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

3.5 Sorting Items

Sitecore sorts the children of each item according to the child sorting rule associated with the item. You can choose a different child sorting rule for each item. Sitecore provides a number of child sorting rules. If you do not specify a child sorting rule for an item, Sitecore applies the Default child sorting rule, and users can sort items manually.

Note
If you apply a child sorting rule to an item or the standard values of its data template, you cannot override that child sorting rule by manually sorting items.

Note
An item comparer can interpret leading digits in item names as characters or as numbers, causing items with names beginning with `10` to sort before or after items with names beginning with `2`.

Note
The names of various system items begin with two underscore characters ("__"). An item comparer can sort items with names with leading underscores to the beginning or end of the list.

3.5.1 Child Sorting Rules

Sitecore provides the following child sorting rules:

Rule	Logic
Created	Sort items by the creation date of the most recent version of each item. Items with oldest versions sort last.
Default	Sort items alphabetically by name, not interpreting leading digits as numbers. Leading underscores sort last. This is the default child sorting rule.
Display Name	Sort items alphabetically by display name, interpreting leading digits as numbers. Leading underscores sort last.
Logical	Sort items alphabetically by name, interpreting leading digits as numbers. Leading underscores sort first.
Reverse	Sort items in reverse alphabetical order by name, not interpreting digits as numbers. Leading underscores sort first.
Updated	Sort items by when each was last updated. Recently updated items sort first.
[Reset to Standard]	Resets the child sorting rule for the item to that defined in the standard values of its data template, or to the default child sorting rule.

3.5.2 How to Select a Child Sorting Rule

To select a child sorting rule to control the sorting of the children of an item:

1. In the **Template Manager** or the **Content Editor**, select the standard values item or the individual item.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

2. On the **Home** tab, in the **Sorting** group, click the child sorting rule dialog launcher (at the bottom of the **Sort** group, the square icon containing an arrow). The child sorting rule dialog appears.
3. In the child sorting rule dialog, select the child sorting rule.

3.5.3 How to Implement a Custom Item Comparer

You can implement a custom item comparer to sort a list of `Sitecore.Data.Items.Item` objects. You can configure the comparer as a child sorting rule, and you can call the comparer from your code. To create a custom item comparer, implement the `Compare()` method of the `System.Collections.Generic.IComparer<Sitecore.Data.Items.Item>` interface.

Example: Sort Items by a Field Value

You can implement a custom item comparer that sorts items by a common field value.

3.5.4 How to Implement a Custom Child Sorting Rule

To implement a custom child sorting rule:

1. Implement a custom item comparer as described in the section *How to Implement a Custom Item Comparer*.

   For example, to sort items by the field named `Title`:

```
namespace Sitecore.Sharedsource.Data.Comparers
{
  public class TitleComparer : Sitecore.Sharedsource.Data.Comparers.ItemFieldComparer
  {
    public TitleComparer() : base("title")
    {
    }
  }
}
```

2. In the **Content Editor**, select the `/Sitecore/System/Settings/Subitems Sorting` item.
3. Insert a child sorting rule definition item using the `/System/Child Sorting` data template. For the name of the item, use the name of the custom item comparer class or the name of the property by which the custom item comparer sorts.
4. In the **Content Editor**, select the child sorting rule definition item.
5. In the new child sorting rule definition item, in the **Data** section, in the **Type** field, enter the signature of the custom item comparer class:

```
Sitecore.Sharedsource.Data.Comparers.TitleComparer, Assembly
```

For more information about selecting the child sorting rule for an item, see the section *How to Select a Child Sorting Rule*.

3.5.5 How to Sort a List of Items

You can use the `Array.Sort()` method with an item comparer to sort a list of items. You can implement a solution based on the following example that sorts the children of the context item using the Title field comparer described in the section How to Implement a Custom Child Sorting Rule:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
Sitecore.Data.Items.Item[] items = contextItem.Children.ToArray();
Sitecore.Sharedsource.Data.Comparers.TitleComparer titleComparer = new
  Sitecore.Sharedsource.Data.Comparers.TitleComparer();
Array.Sort(items, titleComparer);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

You can invoke the `Array.Reverse()` method to reverse the sort order after invoking a custom item comparer as described in the section *Example: Latest Items Web Control*:

```
        Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
        Sitecore.Data.Items.Item[] items = contextItem.Children.ToArray();
        Sitecore.Sharedsource.Data.Comparers.TitleComparer titleComparer = new
          Sitecore.Sharedsource.Data.Comparers.TitleComparer();
        Array.Sort(items, titleComparer);
        Array.Reverse(items);
```

Alternatively, you can implement a custom item comparer based on an existing comparer. For example, to implement a comparer that sorts items in reverse alphabetical order by the value of the Title field:

```
        namespace Sitecore.Sharedsource.Data.Comparers
        {
          public class ReverseTitleComparer :
Sitecore.Sharedsource.Data.Comparers.TitleComparer
          {
            public override int Compare(Sitecore.Data.Items.Item xItem,
              Sitecore.Data.Items.Item yItem)
            {
              return base.Compare(xItem, yItem) * -1;
            }
          }
        }
```

Tip
To use one of the item comparers used by the child sorting rules described in the section Child Sorting Rules, use the class specified in the Type field in the Data section of the corresponding child sorting rule definition item beneath the `/Sitecore/System/Settings/Subitems Sorting` item.

3.5.6 Example: Latest Items Web Control

You can implement a web control based on the following example that accesses the most recent items based on a specific data template that are descendants of the data source item of the rendering:

```
        using System;

        namespace Sitecore.Sharedsource.Web.UI.WebControls
        {
          public class LatestArticles : Sitecore.Web.UI.WebControl
          {
            private string  article Template = null;
            private string  sort Field = null;
            private int _maxArticles = -1;

            public string ArticleTemplate
            {
              get
              {
                return this._article_Template;
              }

              set
              {
                this._article_Template = value;
              }
            }

            public string SortField
            {
              get
              {
                return this. sort Field;
              }

              set
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
            {
              this. sort Field = value;
            }
          }

          public int MaxArticles
          {
            get
            {
              return this. maxArticles;
            }

            set
            {
              this. maxArticles = value;
            }
          }

          protected override void DoRender(System.Web.UI.HtmlTextWriter output)
          {
            Sitecore.Diagnostics.Assert.IsNotNull(
              this.ArticleTemplate,
              "Specify the ArticleTemplate property");
            Sitecore.Diagnostics.Assert.IsNotNullOrEmpty(
              this.SortField,
              "Specify the SortField property");
            Sitecore.Diagnostics.Assert.IsTrue(
              this.MaxArticles > 0,
              "Specify the MaxArticles property");
            Sitecore.Data.Items.Item dataSource = GetItem();
            Sitecore.Diagnostics.Assert.IsNotNull(
              dataSource,
              "Invalid data source item");
            Sitecore.Data.Items.TemplateItem articleTemplate =
              Sitecore.Context.Database.Templates[this.ArticleTemplate];
            Sitecore.Diagnostics.Assert.IsNotNull(
              articleTemplate,
              "Invalid ArticleTemplate " + this.ArticleTemplate);
            string query = String.Format(
              "{0}//*[@@templateid='{1}']",
              dataSource.Paths.FullPath,
              articleTemplate.ID);
            Sitecore.Data.Items.Item[] queried =
Sitecore.Context.Database.SelectItems(query);

            if (!this.WriteItems(queried, output))
            {
              string message = String.Format(
                "{0} : No items under {1} based on {2}",
                GetType(),
                dataSource.Paths.FullPath,
                articleTemplate.FullName);
              Sitecore.Diagnostics.Log.Warn(message, this);
            }
          }

          private bool WriteItems(
            Sitecore.Data.Items.Item[] items,
            System.Web.UI.HtmlTextWriter output)
          {
            if (items != null && items.Length > 0 && MaxArticles > 0)
            {
              Sitecore.Sharedsource.Data.Comparers.ItemFieldComparer comparer =
                new Sitecore.Sharedsource.Data.Comparers.ItemFieldComparer(SortField);
              Array.Sort(items, comparer);
              Array.Reverse(items);

              if (items.Length > this.MaxArticles)
              {
                Array.Resize(ref items, this.MaxArticles);
              }

              output.Write("<ul>");
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
                foreach (Sitecore.Data.Items.Item article in items)
                {
                  string markup = String.Format(
                    @"<li><a href=""{0}"">{1}</a> ({2})</li>",
                    Sitecore.Links.LinkManager.GetItemUrl(article),
                    article.Name,
                    article.Statistics.Created.ToString("f"));
                  output.Write(markup);
                }

                output.Write("</ul>");
                return true;
              }

              return false;
            }
          }
        }
```

The latest articles web control exposes several properties that control its behavior.

The `SortField` property of the latest articles web control contains the name or ID of the field by which to sort. For example, you can specify `__Created` to sort by the version creation date, or `__Updated` to sort by the date of the last update for the language.

For more information about implementing a web control and passing parameters including a data source to a control, see the manual *Presentation Component Cookbook*.

Tip
If you create a Datetime field for sorting, then you can set the standard value of that field to `$now` to set the initial value of the field to date and time of creation of the item.

For more information about entering values such as `$now` in the standard value for a Date or Datetime field, and viewing raw field values, see the manual *Client Configuration Cookbook*.

The `ArticleTemplate` property of the latest articles web control contains the full name of the article data template. For example, to list items based on the `Sample/Sample Item` data template, set `ArticleTemplate` to `Sample/Sample Item`.

The `MaxArticles` property of the latest articles web control contains the maximum number of articles to list. For example, to access the five latest articles, set `MaxArticles` to `5`.

The latest articles web control uses a Sitecore query to retrieve all descendants of the data source item based on the data template specified by the `ArticleTemplate` property (the default data source item for a rendering is the context item). The control then sorts the articles by the field specified by the `SortField` property using the item comparer described in the section Example: Sort Items by a Field Value, reverses the list, resizes the list if it exceeds the limit specified by the `MaxArticles` property, and then renders links to each article.

For more information about specifying a data source for a rendering, see the manual *Presentation Component Cookbook*.

Note
This example uses the descendant axis (“`//`”), which can become inefficient as the number of descendants grows. For solutions that involve a large number of items, consider alternative information architectures to prevent use of the descendant axis. For example, store news articles in folders that represent years and months, and write code to access only the latest folders instead of using the descendant axis.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Chapter 4

Accessing Fields

This chapter provides information about APIs that you can use to access fields. This chapter begins with an overview of accessing fields, and then describes the `renderField` pipeline, the FieldRenderer web control, and the field type web controls, concluding with examples of accessing fields.

This chapter contains the following sections:

- Overview of Accessing Fields
- The renderField Pipeline
- The FieldRenderer Web Control
- The Field Type Web Controls
- Examples of Accessing Fields

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

4.1 Overview of Accessing Fields

You can use the `Sitecore.Data.Items.Item.Fields` collection property to access field values as strings or using specific classes in the `Sitecore.Data.Fields` namespace for different types of fields.

The raw values of Rich Text Editor (RTE) fields can contain Global Unique IDentifiers (GUIDs, or IDs) representing dynamic links from one item to another. Presentation components must use constructs that transform dynamic links in raw RTE field values to friendly URLs.

For more information about dynamic links and friendly URLs, see the *Content API Cookbook*.

Presentation components that render the values of field types that support inline editing must use APIs that generate inline editing controls. You can use the APIs described in this section to transform dynamic URLs and render inline editing controls.

Note
It is not necessary to enable inline editing controls for every use of every field. Use inline editing features only where you want to enable inline editing.

For more information about APIs that access fields, see the manual *Content API Cookbook*.

4.1.1 How to Transform Dynamic Links in RTE Fields to Friendly URLs

You can transform dynamic links in RTE fields to friendly URLs without enabling inline editing using the `Sitecore.Links.LinkManager.GetItemUrl()` method. For example, to transform dynamic links in the Text field in the context item to friendly URLs:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
string containsDynamicLinks = contextItem.Fields["text"].Value;
string containsFriendlyLinks = Sitecore.Links.LinkManager.ExpandDynamicLinks(
  containsDynamicLinks, Sitecore.Configuration.Settings.Rendering.SiteResolving);
string mediaPrefix = string.Empty;
foreach (string currentPrefix in
 Sitecore.Resources.Media.MediaManager.Provider.Config.MediaPrefixes)
{
  if (containsFriendlyLinks.IndexOf(currentPrefix, 0) > 0)
  {
    mediaPrefix = currentPrefix;
    break;
  }
}
if (mediaPrefix.Length > 0)
{
  finalMarkup = System.Text.RegularExpressions.Regex.Replace(containsFriendlyLinks,
    "([^/])" + mediaPrefix, "$1/" + mediaPrefix);
}
else
{
  finalMarkup = containsFriendlyLinks;
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

4.2 The renderField Pipeline

You can use the `renderField` pipeline to transform dynamic links in field values to friendly URLs and to generate inline editing controls for the field types that support them when inline editing in the Page Editor. For example, to render the `Text` field in the context item using the `renderField` pipeline:

```
Sitecore.Pipelines.RenderField.RenderFieldArgs args =
  new Sitecore.Pipelines.RenderField.RenderFieldArgs();
args.Item = Sitecore.Context.Item;
args.FieldName = "text";
Sitecore.Pipelines.CorePipeline.Run("renderField", args);
string markup = args.Result.ToString();
```

The `renderField` pipeline supports the following data template field types:

- Date
- Datetime
- Image
- Integer
- Multi-line Text
- Number
- Rich Text
- Single-line Text

For information about constructs that use the `renderField` pipeline to access field values, see the section The FieldRenderer Web Control and the section The Field Type Web Controls.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

4.3 The FieldRenderer Web Control

You can use the *FieldRenderer* web control (`Sitecore.Web.UI.WebControls.FieldRenderer`) to transform dynamic links in field values to friendly URLs and to generate inline editing controls for the field types that support them when inline editing in the Page Editor. You can use the *FieldRenderer* web control as a web control, or you can use the static

For more information about the *FieldRenderer* web control, see the manual *Content API Cookbook*.

`Sitecore.Web.UI.WebControls.FieldRenderer.Render()` method. For information about the `renderField` pipeline used by the *FieldRenderer* web control, see the section *The renderField Pipeline*.

To use the *FieldRenderer* web control as a web control, add a control such as the following to a layout or sublayout file:

```
        <sc:FieldRenderer runat="server" fieldname="//TODO: field" />
```

You can use the `datasource` attribute of the *FieldRenderer* web control, or of the field type web controls control that inherit from the *FieldRenderer* web control, to specify the item containing the field — the default data source item for a rendering is the context item.

For more information about specifying a data source for a rendering, see the manual *Presentation Component Cookbook*.

You can invoke the static `Sitecore.Web.UI.WebControls.FieldRenderer.Render()` method to transform dynamic links in a field and generate inline editing controls when inline editing in the Page Editor. For example, to generate markup for a Single-line Text field named Title in the context item:

```
        Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
        string markup = Sitecore.Web.UI.WebControls.FieldRenderer.Render(contextItem,
"title");
```

You can add an instance of the *FieldRenderer* web control to the controls collection of another control. You can implement a `Page_Load()` method in a layout or sublayout code file based on the following example that adds a *FieldRenderer* web control for the **Title** field in the context item:

```
        protected void Page_Load(object sender, EventArgs e)
        {
          Sitecore.Web.UI.WebControls.FieldRenderer titleRenderer =
            new Sitecore.Web.UI.WebControls.FieldRenderer();
          titleRenderer.FieldName = "title";
          this.Controls.Add(titleRenderer);
        }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

4.4 The Field Type Web Controls

You can use the field type web controls described in this section to transform dynamic links in specific types of fields to friendly URLs and to generate inline editing controls in the Page Editor for the field types that support them. The field type web controls inherit from the FieldRenderer web control. For more information about the FieldRenderer web control, see the section The FieldRenderer Web Control.

For more information about the *FieldRenderer* web control, see the manual *Content API Cookbook*.

4.4.1 The Link Field Web Control

You can use the Link field web control (`Sitecore.Web.UI.WebControls.Link`) to generate an HTML anchor ("`<a>`") element based on the value in a field of type General Link. For example, to use the Link field web control to render the General Link field named GeneralLinkField in the context item, add a Link field web control such as the following to a layout or sublayout file:

```
        <sc:link field="generallinkfield" runat="server" />
```

You can use the `text` attribute of the Link field web control to override the text of the link. For example, to use the Link field web control to render the General Link field named GeneralLinkField in the context item, overriding the text of the link:

```
        <sc:link field="generallinkfield" runat="server" text="//TODO: replace with link text"
/>
```

Alternatively, you can enclose text within the Link field web control to override the text of the link:

```
        <sc:link field="generallinkfield" runat="server">
          //TODO: replace with link text
        </sc:link>
```

Note
The the Link field web control can include markup elements such as HTML image elements and other ASP.NET controls.

4.4.2 The Image Field Web Control

You can use the Image field web control (`Sitecore.Web.UI.WebControls.Image`) to generate an HTML image element (`<img>`) based on a field of type Image. For example, to use the Image field web control to process the Image field named ImageField in the context item, add an Image field web control such as the following to a layout or sublayout file:

```
        <sc:image field="imagefield" runat="server" />
```

You can use additional attributes to specify additional image properties. For example, to use the Link field web control with the `alt` attribute to override the alternate text for the Image field named ImageField in the context item:

```
        <sc:image field="imagefield" runat="server" alt="//TODO: alternate text" />
```

4.4.3 The Date Field Web Control

You can use the Date field web control (`Sitecore.Web.UI.WebControls.Date`) to render a string representation of a field of type Date or Datetime. For example, to use the Date field web control to render the Datetime field named `DatetimeField` in the context item, add a Date field web control such as the following to a layout or sublayout file:

```
        <sc:date field="datetimefield" runat="server" />
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

You can use the `format` attribute of the Date field web Control to specify a .NET date and time format pattern. For example, to use the Date field web control to render the Datetime field named DatetimeField in the context item using the full date and time pattern with short time:

```
        <sc:date field="datetimefield" runat="server" format="f" />
```

For more information about .NET date and time format patterns, see http://msdn.microsoft.com/en-us/library/97x6twsz(VS.71).aspx

4.4.4 The Text Field Web Control

You can use the Text field web control (`Sitecore.Web.UI.WebControls.Text`) to render a field of type Multi-line Text, Rich Text, or Single-line Text. For example, to use the Text field web control to render the Single-Line Text field named in the context item, add a Text field web control such as the following to a layout or sublayout file:

```
        <sc:text field="singlelinetextfield" runat="server" />
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

4.5 Examples of Accessing Fields

This section contains examples that access different types of fields.

4.5.1 Example: Link to Item in a Selection Field

You can implement presentation logic based on the following example that links to the item selected in the field of type Droplink, Droptree, or Grouped Droplink named SelectionField in the context item:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
Sitecore.Data.Fields.ReferenceField selectionField =
  contextItem.Fields["selectionfield"];

if (selectionField!=null && selectionField.TargetItem!=null)
{
  Sitecore.Data.Items.Item target = selectionField.TargetItem;
  string markup = String.Format(
    @"<a href=""{0}"">{1}</a>",
    Sitecore.Links.LinkManager.GetItemUrl(target),
    target.Name);
}
```

4.5.2 Example: Link to Media in a File or Image Field

You can implement presentation logic based on the following example that links to the media item selected in the field of type File named FileField in the context item:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
Sitecore.Data.Fields.FileField fileField = contextItem.Fields["filefield"];

if (fileField != null && fileField.MediaItem != null)
{
  Sitecore.Data.Items.MediaItem media =
    new Sitecore.Data.Items.MediaItem(fileField.MediaItem);
  string markup = string.Format(
    @"<a href=""{0}"">{1}</a>",
    Sitecore.Resources.Media.MediaManager.GetMediaUrl(media),
    media.Name);
}
```

Note
You can use the same approach to process the image selected in an Image field — use the `Sitecore.Data.Fields.ImageField` class instead of the `Sitecore.Data.Fields.FileField` class.

4.5.3 Example: Link to Items in a Multiselect Field

You can implement presentation logic based on the following example that links to the items selected in the field of type Checklist, Multilist, Treelist, or TreelistEx named MultiselectField in the context item:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
Sitecore.Data.Fields.MultilistField multiselectField =
  contextItem.Fields["multiselectfield"];

if (multiselectField != null && multiselectField.GetItems().Length>0)
{
  System.Text.StringBuilder markupBuilder = new System.Text.StringBuilder("<ul>");

  foreach (Sitecore.Data.Items.Item item in multiselectField.GetItems())
  {
    markupBuilder.Append(string.Format(
      @"<li><a href=""{0}"">{1}</a>",
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
            Sitecore.Links.LinkManager.GetItemUrl(item),
            item.Name));
        }

        markupBuilder.Append("</ul>");
        string markup = markupBuilder.ToString();
        output.Write(markup);
      }
```

4.5.4 Example: Link to Media Items in File Drop Area Field

You can use this example to implement a web control that links to the items in a File Drop Area field:

```
namespace Sitecore.Sharedsource.Web.UI.WebControls
{
  using System;
  using System.Web.UI;

  public class FDAList : Sitecore.Web.UI.WebControl
  {
    public string FieldName
    {
      get;
      set;
    }

    protected override void DoRender(HtmlTextWriter output)
    {
      if (String.IsNullOrEmpty(this.FieldName)
        || Sitecore.Context.Item == null)
      {
        return;
      }

      Sitecore.Data.Fields.FileDropAreaField fdaField =
        Sitecore.Context.Item.Fields[this.FieldName];

      if (fdaField == null
        || String.IsNullOrEmpty(fdaField.Value)
        || fdaField.MediaFolder == null)
      {
        return;
      }

      Sitecore.Collections.ChildList children = fdaField.MediaFolder.Children;

      if (children.Count < 1)
      {
        return;
      }

      foreach (Sitecore.Data.Items.Item media in fdaField.MediaFolder.Children)
      {
        string url = Sitecore.StringUtil.EnsurePrefix(
          '/',
          Sitecore.Resources.Media.MediaManager.GetMediaUrl(media));
        string message = String.Format(
          @"<a href=""{0}"">{1}</a>",
          url,
          media.Name);
        output.WriteLine(message);
      }
    }
  }
}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Chapter 5

Working with Languages

This chapter contains information about APIs that you can use to access item data in multiple languages. This chapter contains an overview of language processing with Sitecore, followed by several examples.

This chapter contains the following sections:

- Overview
- Examples

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

5.1 Overview of Languages

Each item can contain multiple languages. Each language can contain multiple versions. Unless otherwise specified, Sitecore access field values from the latest version of each item in the context language. For more information about the context language, see the section *The Context Language*.

Important
If an item does not contain version data for a language, all versioned field values for that item are empty for that language.

Note
The value of a shared field applies to all versions in all languages. An item with no version data for a language contains shared field values for that language.

Important
Sitecore does not automatically fall back to an alternate language when no field value exists for the context language. For suggestions to support fallback languages, see the section Supporting Fallback Languages.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

5.2 Examples Involving Languages

This section provides examples of APIs involving languages.

5.2.1 Example: Link to Items with Version Data for the Context Language

You can use the `Sitecore.Data.Items.Item.Versions.Count` property to determine whether version data exists for a language of an item. You can develop a solution based on the following example that generates a list of links to each child of the context item that contains version data in the context language and has layout details for the context device:

```
System.Text.StringBuilder markup = new System.Text.StringBuilder();
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;

foreach (Sitecore.Data.Items.Item child in contextItem.Children)
{
  if (child.Versions.Count > 0
    && child.Visualization.GetLayout(Sitecore.Context.Device) != null)
  {
    string url = Sitecore.Links.LinkManager.GetItemUrl(child);
    markup.AppendFormat(@"<li><a href=""{0}"">{1}</a></li>", url, child.Name);
  }
}

if (markup.Length > 0)
{
  string list = "<ul>" + markup.ToString() + "</ul>";
}
```

5.2.2 Example: Language Flags Web Control

You can implement a web control based on the following example that renders flags linking to alternate languages of the context item:

```
using System;

namespace Sitecore.Sharedsource.Web.UI.WebControls
{
  public class LanguageFlags : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(System.Web.UI.HtmlTextWriter output)
    {
      string spacer = " ";
      System.Text.StringBuilder flags = new System.Text.StringBuilder();

      foreach (Sitecore.Globalization.Language language in
        Sitecore.Context.Database.Languages)
      {
        if (language == Sitecore.Context.Language)
        {
          flags.Append(this.GetLanguageFlag(language) + spacer);
        }
        else
        {
          Sitecore.Data.Items.Item langItem =
            Sitecore.Context.Database.GetItem(Sitecore.Context.Item.ID, language);

          if (langItem.Versions.Count > 0)
          {
            string markup = String.Format(
              @"<a href=""{0}"">{1}</a>{2}",
              this.GetLanguageUrl(langItem),
              this.GetLanguageFlag(language),
              spacer);
            flags.Append(markup);
          }
        }
      }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
            if (flags.Length > 0)
            {
              flags.Length = flags.Length - spacer.Length;
              output.Write(flags);
            }
          }

          private string GetLanguageFlag(Sitecore.Globalization.Language language)
          {
            string icon = language.GetIcon(Sitecore.Context.Database);

            if (icon.StartsWith("~"))
            {
              icon = Sitecore.StringUtil.EnsurePrefix('/', icon);
            }
            else if ((!icon.StartsWith("/")) && (!icon.Contains(":")))
            {
              icon = Sitecore.Resources.Images.GetThemedImageSource(icon);
            }

            return String.Format(
              @"<img src=""{0}"" alt=""{1}"" />",
              icon,
              language.GetDisplayName());
          }

          private string GetLanguageUrl(Sitecore.Data.Items.Item item)
          {
            Sitecore.Links.UrlOptions urlOptions =
              Sitecore.Links.LinkManager.GetDefaultUrlOptions();
            urlOptions.Language = item.Language;
            urlOptions.LanguageEmbedding = Sitecore.Links.LanguageEmbedding.Always;
            return Sitecore.Links.LinkManager.GetItemUrl(item, urlOptions);
          }
        }
      }
```

The language flags web control iterates over each language in the context database. If that language is the same as the context language, then the control displays the flag associated with that language. Otherwise, if a version of the context item exists in that language, the control displays the flag associated with that language as a link to the context item in that language.

For instructions to implement a web control, see the *Presentation Component Cookbook*.

5.2.3 Example: Determine Context Language from the User Profile

You can implement a pipeline processor based on the following example that determines the context language from the profile of the context user if the cookie, query string, and URL path do not specify a language:

```
      namespace Sitecore.Sharedsource.Pipelines.HttpRequest
      {
        public class ProfileLanguageResolver :
          Sitecore.Sharedsource.Pipelines.HttpRequest.LanguageProcessor
        {
          protected override bool IsLanguageKnown()
          {
            return this.QueryHasLanguage()
              || this.PathHasLanguage()
              || this.CookieHasLanguage();
          }

          public void Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
          {
            if ((!this.IsLanguageKnown()) && this.ProfileHasLanguage())
            {
              Sitecore.Globalization.Language language;

              if (Sitecore.Globalization.Language.TryParse(
                Sitecore.Context.User.Profile.RegionalIsoCode, out language))
              {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
                Sitecore.Context.SetLanguage(language, true);
                Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;

                if (contextItem != null)
                {
                  Sitecore.Context.Item =
                    Sitecore.Context.Database.GetItem(contextItem.ID, language);
                }
              }
            }
          }
        }
```

Insert the processor after the item resolver within the `/configuration/sitecore/pipelines/httpRequestBegin` element in `web.config`:

```
      <httpRequestBegin>
      ...
        <processor type="Sitecore.Pipelines.HttpRequest.ItemResolver, Sitecore.Kernel" />
        <processor type=
          "Sitecore.Sharedsource.Pipelines.HttpRequest.ProfileLanguageResolver, Assembly" />
      ...
```

The `Sitecore.Sharedsource.Pipelines.HttpRequest.ProfileLanguageResolver.Process()` method sets the context language to the language specified in the user's profile if the user is not anonymous and has a profile language.

The Language Processor Base Class

The language processor base class `Sitecore.Sharedsource.Pipelines.HttpRequest.LanguageProcessor` provides the `QueryHasLanguage()`, `PathHasLanguage()`, `CookieHasLanguage()`, and `ProfileHasLanguage()` methods that indicate which properties of the HTTP request contain language information. Processors classes that inherit from this base class must override the `IsLanguageKnown()` method, which indicates whether criteria with greater priority determine the context language.

```
      using System;

      namespace Sitecore.Sharedsource.Pipelines.HttpRequest
      {
        public abstract class LanguageProcessor
        {
          protected bool QueryHasLanguage()
          {
            return !String.IsNullOrEmpty(Sitecore.Web.WebUtil.GetQueryString("sc lang"));
          }

          protected bool PathHasLanguage()
          {
            return Sitecore.Context.Data.FilePathLanguage != null;
          }

          protected bool CookieHasLanguage()
          {
            if (Sitecore.Context.Site != null)
            {
              string cookie = Sitecore.Web.WebUtil.GetCookieValue(
                Sitecore.Context.Site.GetCookieKey("lang"));

              if (!String.IsNullOrEmpty(cookie))
              {
                return true;
              }
            }

            return false;
          }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
        protected bool ProfileHasLanguage()
        {
          return Sitecore.Context.IsLoggedIn &&
            !String.IsNullOrEmpty(Sitecore.Context.User.Profile.RegionalIsoCode);
        }

        protected abstract bool IsLanguageKnown();
      }
    }
```

Note
If processors between the language resolver and the item resolver depend on the context language, then insert the custom language processor immediately after the language resolver. Otherwise, if you insert a custom processor after the item resolver, then that processor can use the `Sitecore.Data.Items.Items.Versions.Count` property to determine which languages contain version data for each language. Insert the processor before the culture resolver described in the section Example: Set Thread Culture.

5.2.4 Example: Determine Language from Web Client Preferences

You can implement a pipeline processor based on the following example that determines the context language from web client preferences:

```
    using System;

    namespace Sitecore.Sharedsource.Pipelines.HttpRequest
    {
      public class ClientLanguageResolver : LanguageProcessor
      {
        protected override bool IsLanguageKnown()
        {
          return QueryHasLanguage() || PathHasLanguage() || CookieHasLanguage() ||
            ProfileHasLanguage();
        }

        public void Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
        {
          if (!this.IsLanguageKnown())
          {
            string langs = args.Context.Request.ServerVariables["HTTP ACCEPT LANGUAGE"];

            if (!String.IsNullOrEmpty(langs))
            {
              foreach (string requested in Sitecore.StringUtil.Split(langs, ',', true))
              {
                Sitecore.Globalization.Language language;
                string languageCode = requested;

                if (languageCode.IndexOf(';') > -1)
                {
                  languageCode = languageCode.Substring(0, languageCode.IndexOf(';'));
                }

                Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;

                if (Sitecore.Globalization.Language.TryParse(languageCode, out language)
                  && (contextItem == null
                  || contextItem.Versions.Count > 0))
                {
                  Sitecore.Context.SetLanguage(language, true);

                  if (contextItem != null)
                  {
                    Sitecore.Context.Item =
                      Sitecore.Context.Database.GetItem(contextItem.ID, language);
                  }

                  return;
                }
              }
            }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
            }
          }
        }
      }
```

Insert the processor after the language resolver within the `/configuration/sitecore/pipelines/httpRequestBegin` element in `web.config`:

```
      <httpRequestBegin>
      ...
        <processor type="Sitecore.Pipelines.HttpRequest.LanguageResolver, Sitecore.Kernel"
/>
        <processor type=
          "Sitecore.Sharedsource.Pipelines.HttpRequest.ClientLanguageResolver, Assembly" />
      ...
```

The `Sitecore.Sharedsource.Pipelines.HttpRequest.ClientLanguageResolver.Process()` method parses the `HTTP_ACCEPT_LANGUAGE` header variable sent in the HTTP request from the web client. The method sets the context language to the first specified language if the context item is unknown, or the first language for which the context item contains version data. For information about the base class `Sitecore.Sharedsource.Pipelines.HttpRequest.LanguageProcessor`, see the section *The Language Processor Base Class*.

Note
If you do not set the context language from the user profile as described in the section Example: Determine Context Language from the User Profile, remove the `ProfileHasLanguage()` method and update the `IsLanguageKnown()` method accordingly.

Note
See the note regarding the context language in the previous section, The Language Processor Base Class.

5.2.5 Example: Set Thread Culture

ASP.NET does not automatically format dates and numbers according to the region code associated with the context language. You can implement a custom processor based on the following example that sets the culture for the thread to control the formatting of dates and numbers:

```
      namespace Sitecore.Sharedsource.Pipelines.HttpRequest
      {

        public class CultureResolver
        {
          public void Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
          {
            if (!Sitecore.Context.Language.CultureInfo.IsNeutralCulture)
            {
              System.Threading.Thread.CurrentThread.CurrentUICulture =
                Sitecore.Context.Language.CultureInfo;
              System.Threading.Thread.CurrentThread.CurrentCulture =
                Sitecore.Context.Language.CultureInfo;
            }
          }
        }
      }
```

Insert the culture resolver after the language resolver within the `/configuration/sitecore/pipelines/httpRequestBegin` element in `web.config`:

```
      <httpRequestBegin>
      ...
        <processor type="Sitecore.Pipelines.HttpRequest.LanguageResolver, Sitecore.Kernel"
/>
        <processor
          type="Sitecore.Sharedsource.Pipelines.HttpRequest.CultureResolver, Assembly" />
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
...
```

Note
ASP.NET automatically resets the thread culture after processing the HTTP request.

Note
Insert the culture resolver after any custom language resolvers such as those described in these sections: *Example: Determine Context Language from the User Profile*, *Example: Determine Language from Web Client Preferences*, *Example: Item Fallback Languages*.

5.2.6 Supporting Fallback Languages

You can use the following procedure to support fallback languages in multilingual solutions, allowing you to display a value from an alternate language if a field or item does not have a value for the context language:

1. In the **Template Manager** or the **Content Editor**, create a custom data template named **Custom Language**.

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

2. Add a section named **Custom** to the **Custom Language** data template.
3. Add a **Droplist** field named **Fallback Language** in the **Custom** section of the **Custom Language** data template.
4. Select **Shared** for the **Fallback Language** field in the **Custom** section of the **Custom Language** data template.
5. Set the **Source** property of the **Fallback Language** field in the **Custom** section of the **Custom Language** data template to `/sitecore/system/languages`.
6. Configure base templates for the `System/Language` data template to include the **Custom Language** template.
7. In the **Content Editor**, under the `/Sitecore/System/Languages` item, in each language definition item, in the **Custom** section, in the **Fallback Language** field, select a fallback language.
8. In the Visual Studio web application project, implement an extension method for the `Sitecore.Globalization.Language` class providing access to the fallback language for a language:

```
using System;

namespace Sitecore.Sharedsource.Globalization
{
  public static class LanguageExtensions
  {
    private const string FALLBACK LANGUAGE FIELD = "fallback language";

    public static Sitecore.Globalization.Language GetFallbackLanguage(
      this Sitecore.Globalization.Language language)
    {
      if (Sitecore.Context.Database != null)
      {
        Sitecore.Data.ID languageId =
          Sitecore.Data.Managers.LanguageManager.GetLanguageItemId(
          language,
          Sitecore.Context.Database);

        if (languageId != (Sitecore.Data.ID) null)
        {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
                Sitecore.Data.Items.Item languageItem =
                  Sitecore.Context.Database.GetItem(languageId);

                if (languageItem != null)
                {
                  string fallbackLanguageName = languageItem[FALLBACK_LANGUAGE_FIELD];

                  if (!String.IsNullOrEmpty(fallbackLanguageName))
                  {
                    return Sitecore.Globalization.Language.Parse(fallbackLanguageName);
                  }
                }
              }
            }

            return null;
          }
        }
      }
```

Note
Any code that uses this extension must explicitly reference the `Sitecore.Sharedsource.Globalization` namespace:

```
using Sitecore.Sharedsource.Globalization;
```

You can use the `Sitecore.Globalization.Language.GetFallbackLanguage()` extension method to access the fallback language associated with a language. For example, to access the fallback language for the context language:

```
using Sitecore.Sharedsource.Globalization;
...
Sitecore.Globalization.Language contextLanguage = Sitecore.Context.Language;
Sitecore.Globalization.Language fallbackLanguage =
   contextLanguage.GetFallbackLanguage();
```

Note
An item may contain version data for a language without containing a value for each field.

Example: Field Fallback Languages

If a field does not have a value in the context language, you can retrieve the corresponding field value from a fallback language. For instructions to support fallback languages, see the section *Supporting Fallback Languages*. You can implement logic based on the following example that retrieves the value of the Title field from the context item in the context language or the first fallback language with a value for that field:

```
using Sitecore.Sharedsource.Globalization;
...
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
Sitecore.Data.Fields.Field title = contextItem.Fields["title"];
string markup = String.Empty;

if (!String.IsNullOrEmpty(title.Value) || Sitecore.Context.PageMode.IsPageEditor)
{
  markup = Sitecore.Web.UI.WebControls.FieldRenderer.Render(contextItem, "title");
}
else
{
  Sitecore.Globalization.Language language = Sitecore.Context.Language;

  do
  {
    language = language.GetFallbackLanguage();

    if (language != null)
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
            {
              Sitecore.Data.Items.Item languageItem =
                contextItem.Database.GetItem(contextItem.ID, language);
              title = languageItem.Fields[title.ID];
            }
          }
          while (language != null && String.IsNullOrEmpty(title.Value));

          if (!String.IsNullOrEmpty(title.Value))
          {
            markup = title.Value;
          }
        }
```

Important
Do not provide inline editing controls for field values from languages other than the context language except in the Page Editor.

Example: Item Fallback Languages

If the context item does not contain version data for the context language, you can set the context language to a fallback language. For instructions to support fallback languages, see the section Supporting Fallback Languages. You can implement a pipeline processor based on the following example that sets the context language to a fallback language if the context item does not contain version data for the context language:

```
        public class FallbackLanguageProcessor
            {
                public void Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
                {
                    Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
                    if (contextItem == null || contextItem.Versions.Count > 0)
                    {
                        return;
                    }
                    Sitecore.Globalization.Language contextLanguage =
                        Sitecore.Context.Language;

                    Sitecore.Globalization.Language fallbackLanguage =
                        contextLanguage.GetFallbackLanguage();
                    if (fallbackLanguage == null)
                    {
                        return;
                    }
                    if (fallbackLanguage.Name.Equals(contextLanguage.Name,
                        StringComparison.InvariantCultureIgnoreCase))
                    {
                        return;
                    }

                    Sitecore.Data.Database contextDatabase = Sitecore.Context.Database;
                    Sitecore.Data.Items.Item fallbackItem =
                        contextDatabase.GetItem(Sitecore.Context.Item.ID, fallbackLanguage);
                    if (fallbackItem != null && fallbackItem.Versions.Count > 0)
                    {
                        Sitecore.Context.Item = fallbackItem;
                        Sitecore.Context.Language = fallbackLanguage;
                    }
                }
            }
```

Note
This sample code provides a very simple way to implement language fallback in Sitecore which may not be sufficient for complex solutions. See the Language Fallback Item Provider module in the shared source library for additional detail.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Insert the fallback language resolver after the item resolver in the `httpRequestBegin` pipeline in `web.config`:

```
        <httpRequestBegin>
        ...
          <processor type="Sitecore.Pipelines.HttpRequest.ItemResolver, Sitecore.Kernel" />
          <processor type=
            "Sitecore.Sharedsource.Pipelines.HttpRequest.FallbackLanguageProcessor, Assembly"
/>
        ...
```

The fallback language processor does nothing if the context item is unknown, does not exist, has access rights that prevent read access by the context user, or if the context item contains version data for the context language. Otherwise, if a fallback language exists for the context language, and the item contains version data for that language, then set that language as the context item and update the context item to that language.

Note
Insert the fallback language resolver after any other custom language resolvers such as those described in the section *Example: Determine Context Language from the User Profile* and in the section *Example: Determine Language from Web Client Preferences*.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

Chapter 6

Error Management

This chapter provides information that you can use to manage errors and exceptions that can occur in presentation components. Beginning with an overview of exception management, this chapter continues to describe C# `try`/`catch` blocks and Sitecore error pages, and continues to explain how to trap exceptions in renderings, layouts, and applications.

This chapter contains the following sections:

- Exception Management Overview
- Try/Catch Blocks
- Sitecore Error Pages
- How to Trap XSL Rendering Exceptions
- How to Trap Web Control Rendering Exceptions
- How to Trap Layout Exceptions
- How to Trap Application Exceptions

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

6.1 Exception Management Overview

Exceptions represent instances of specific types of error conditions that terminate the current code block, typically when a method cannot complete a task. Exceptions rise through the call stack until they reach a level that handles that type of exception. ASP.NET provides additional facilities that you can use to manage exceptions thrown by presentation components.

In addition to automatically logging all exceptions encountered, Sitecore provides facilities to handle syntax errors and exceptions in XSL renderings and web controls, as well as custom error pages for specific error conditions.

If an XSL rendering contains a syntax error or calls an XSL extension control or method that generates an exception, Sitecore logs and displays information about the XSL rendering and the error. If any other presentation component generates an exception, Sitecore logs that exception and ASP.NET processes the exception normally.

For more information about ASP.NET exception management, see:

- http://msdn.microsoft.com/en-us/library/ms229014(VS.80).aspx
- http://msdn.microsoft.com/en-us/library/ms954830.aspx
- http://msdn.microsoft.com/en-us/library/aa479319.aspx
- http://msdn.microsoft.com/en-us/library/w16865z6.aspx

6.1.1 ASP.NET Exception Management

ASP.NET processes exceptions according to the `/configuration/system.web/customErrors` element in `web.config`.

If the value of the `mode` attribute of the `/configuration/system.web/customErrors` element in `web.config` is `RemoteOnly`, then ASP.NET returns full exception details for HTTP requests that originate on the web server, but redirects or returns a generic error message to all other web clients.

If the value of the `mode` attribute of the `/configuration/system.web/customErrors` element in `web.config` is `Off`, then ASP.NET returns the exception message and full stack trace to the web client as HTML. Exposing these exception details represents a potential security risk.

If the value of the `mode` attribute of the `/configuration/system.web/customErrors` element in `web.config` is `On`, then ASP.NET redirects the web client to the URL specified by the `redirect` attribute of the enclosed `<error>` element with `statusCode` of `500`. For example, given the following configuration, ASP.NET will redirect the web client to `/errors/500.htm` if a presentation component throws an exception:

```
<customErrors mode="On" defaultRedirect="/errors/generic.htm">
  <error statusCode="500" redirect="/errors/500.htm" />
</customErrors>
```

Note
To indicate the page that generated the error, ASP.NET adds the `aspxerrorpath` query string parameter to the URL.

Note
You can add `<error>` elements to redirect the web client to different pages under different error conditions.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

If no `<error>` element with `statusCode` of `500` exists, ASP.NET redirects the web client to the URL specified by the `defaultRedirect` attribute of the `customErrors` element. For example, given the following configuration, ASP.NET will redirect the web client to `/errors/generic.htm` if a presentation component throws an exception:

```
        <customErrors mode="On" defaultRedirect="/errors/generic.htm" />
```

Note
To indicate the page that generated the error, ASP.NET adds the `aspxerrorpath` query string parameter to the URL.

If the `defaultRedirect` attribute is not present in the `<customErrors>` element, then ASP.NET returns a generic HTML error message to the web client.

Tip
Set the `mode` attribute of the `/configuration/system.web/customErrors` element in `web.config` to `On` to test exception management logic. Otherwise, set the `mode` attribute to `Off` or `RemoteOnly`.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

6.2 Try/Catch Blocks

You can use C# `try`/`catch` blocks to trap exceptions locally. For example, the following code logs the exception thrown by an attempted division operation:

```
        int numerator = 5;
        int denominator = 0;

        try
        {
          int quotient = numerator/denominator;

          //TODO: additional logic
        }
        catch (Exception exception)
        {
          string message = String.Format("Unable to divide {0} by {1}", numerator,
denominator);
          Sitecore.Diagnostics.Log.Error(message, exception, this);
        }

        //TODO: additional logic
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

6.3 Sitecore Error Pages

The `value` attributes of the `/configuration/sitecore/settings/setting` elements in `web.config` specify the URLs that Sitecore activates under various error conditions as specified in the following table.

Setting	Error Condition
`ErrorPage`	An HTTP request has raised a known error condition for which there is no specific error handler.
`ItemNotFound`	The URL does not correspond to an item in the database.
`LayoutNotFoundUrl`	The context item does not have valid layout details for the context device.
`NoAccessUrl`	The context user does not have read access to the context item.
`NoLicenseUrl`	Sitecore is not properly licensed.

Note
The `Sitecore.Configuration.Settings` class exposes properties that correspond to these settings.

Note
Depending on configuration, ASP.NET does not always process every HTTP request received by IIS. Sitecore does not process every HTTP request processed by ASP.NET. For consistency, you can set the Sitecore error page settings in `web.config`, the ASP.NET error pages within the `/configuration/system.web/customErrors` element in `web.config`, and the IIS error pages in the IIS management console to the same values.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

6.4 How to Trap XSL Rendering Exceptions

Sitecore logs and clears all exceptions raised by XSL renderings, and embeds information about the exception in the content sent to the web client.

XSL renderings do not support compile-time error detection. Sitecore compiles each XSL rendering when the system first invokes it after an application restart. If the layout engine encounters a syntax error in an XSL rendering at runtime, Sitecore raises an exception before the rendering can generate any output.

XSL extensions controls and methods can throw exceptions. An XSL rendering that does not contain a syntax error can generate output before invoking an XSL extension method or control that generates an exception.

You can trap exceptions and syntax errors in XSL renderings using an error control. When the layout engine encounters a syntax error or exception in an XSL rendering, it invokes the control specified by the `type` attribute of the `/configuration/ErrorControl` element in the `/app_config/prototypes.config` file. The default `Sitecore.Web.UI.WebControls.StandardErrorControl` renders information about the error.

Note
Output generated by a rendering before the exception appears before the output of the error control.

To implement a custom error control, create a class that inherits from `Sitecore.Web.UI.WebControls.ErrorControl` and implement the `Clone()` method. Access the error message and stack trace using the `Message` and `Details` properties of the base class. For example, you can implement an error control based on the following example that, if an XSL rendering contains a syntax error or invokes a .NET XSL extension control or method that throws an exception, depending on the `mode` attribute of the `/configuration/system.web/customErrors` element in `web.config`, either redirects the web client to a friendly error page, or renders information about the error:

```
using System;
using System.Web.UI;

namespace Sitecore.Sharedsource.Web.UI.WebControls
{
  public class ErrorControl : Sitecore.Web.UI.WebControls.StandardErrorControl
  {
    public override Sitecore.Web.UI.WebControls.ErrorControl Clone()
    {
      return (Sitecore.Web.UI.WebControls.ErrorControl) MemberwiseClone();
    }

    protected override void DoRender(HtmlTextWriter output)
    {
      Sitecore.Diagnostics.Log.Error(this.Message, this);
      ErrorHelper helper = new ErrorHelper();
      if (helper.ShouldRedirect())
      {
        helper.Redirect();
      }
      else
      {
        base.DoRender(output);
      }
    }
  }
}
```

The `DoRender()` method of the error control logs the error, and then uses the following helper class to determine to whether to redirect the web client or call the `DoRender()` method in the base class to render details about the error:

```
using System;
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
        using System.Configuration;
        using System.Web.Configuration;

        namespace Sitecore.Sharedsource.Web.UI
        {
          public class ErrorHelper
          {
            private CustomErrorsSection  config = null;
            private string  error500url = null;

            public CustomErrorsSection Config
            {
              get
              {
                if ( config == null)
                {
                  Configuration config =
                    WebConfigurationManager.OpenWebConfiguration("/");
                  this._config =
                    (CustomErrorsSection)config.GetSection("system.web/customErrors");
                  Sitecore.Diagnostics.Assert.IsNotNull(this. config, "customErrors");
                }

                return this. config;
              }
            }

            public string Error500Url
            {
              get
              {
                if (this. error500url == null && this.Config != null)
                {
                  CustomError error500 =
                    this.Config.Errors.Get(500.ToString(
                    System.Globalization.CultureInfo.InvariantCulture));

                  if (error500 != null && !String.IsNullOrEmpty(error500.Redirect))
                  {
                    this._error500url = error500.Redirect;
                  }

                  if (string.IsNullOrEmpty(this. error500url))
                  {
                    this._error500url = Config.DefaultRedirect;
                  }

                  if (this. error500url == null)
                  {
                    this._error500url = String.Empty;
                  }
                }

                return this. error500url;
              }
            }

            public bool ShouldRedirect()
            {
              if (this.Config == null
                || this.Config.Mode == System.Web.Configuration.CustomErrorsMode.On
                    || (this.Config.Mode ==
System.Web.Configuration.CustomErrorsMode.RemoteOnly
                        && !System.Web.HttpContext.Current.Request.IsLocal))
              {
                return true;
              }

              return false;
            }

            public void Redirect()
            {
              if (!String.IsNullOrEmpty(Error500Url))
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

```
            {
              Sitecore.Web.WebUtil.Redirect(
                this.Error500Url + "?aspxerrorpath="
                + Sitecore.Web.WebUtil.GetLocalPath(Sitecore.Context.RawUrl));
            }
            else
            {
              Sitecore.Web.WebUtil.RedirectToErrorPage(
                "//TODO: replace with friendly error message");
            }
          }
        }
      }
```

For more information about implementing a web control, see the Presentation Component Cookbook.

Note
The `Sitecore.Web.WebUtil.RedirectToErrorPage()` method redirects the web client to the page specified by the `value` attribute of the `/configuration/sitecore/settings/setting` element in `web.config` with `name` `ErrorPage`, passing the message specified by the parameter as the URL parameter named `error`.

6.4.1 The Error Helper

The `Sitecore.Sharedsource.Web.UI.ErrorHelper` class provides a method intended to redirect the web client to a friendly error page. The `ShouldRedirect()` method returns True if the `mode` attribute of the `/configuration/system.web/customErrors` section of `web.config` is `On` or is `RemoteOnly` and the current HTTP request is not local. The `Redirect()` method redirects the web client to the first defined value from the following list, adding appropriate query string parameters:

- The page specified by the `/configuration/system.web/customErrors/error` element in `web.config` with `statusCode` of `500`
- The `defaultRedirect` attribute of the `/configuration/system.web/customErrors` element in `web.config`.
- The `value` attribute of the `/configuration/sitecore/settings/setting` element in `web.config` with name `ErrorPage`.

Note
You do not need to register the error control with Sitecore.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

6.5 How to Trap Web Control Rendering Exceptions

Sitecore logs exceptions thrown by web controls before ASP.NET processes them as described in the section Exception Management Overview. You can implement web controls based on the following example that traps exceptions using an error control as described in the section How to Trap XSL Rendering Exceptions:

```
using System;
using System.Web.UI;

namespace Sitecore.Sharedsource.Web.UI.WebControls
{
  public class ExceptionThrower : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(HtmlTextWriter output)
    {
      try
      {
        throw new System.NotImplementedException();
      }
      catch (Exception ex)
      {
        Sitecore.Diagnostics.Log.Error(
          "Web control exception in " + GetType().ToString(), ex, this);
        RenderError(ex.Message, ex.ToString(), output);
      }
    }
  }
}
```

The `DoRender()` method of the web control logs any exceptions, and then invokes the error control described in the section How to Trap XSL Rendering Exceptions for any further error processing.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

6.6 How to Trap Layout Exceptions

You can implement a solution based on the following example that uses the `Page_Error()` method in the code file for a layout to trap exceptions not trapped at a lower level and redirect to a friendly error page:

```
private void Page Error(object sender, EventArgs e)
{
  string message = String.Format(
    "Layout exception in {0} processing {1}",
    GetType().ToString(),
    Sitecore.Context.RawUrl);
  Sitecore.Diagnostics.Log.Error(message, Server.GetLastError(), this);
  Sitecore.Sharedsource.Web.UI.ErrorHelper helper =
    new Sitecore.Sharedsource.Web.UI.ErrorHelper();

  if (helper.ShouldRedirect())
  {
    helper.Redirect();
  }
  else
  {
    Server.ClearError();
  }
}
```

The `Page_Error()` method logs the exception, and then uses the error helper described in the section *The Error Helper* for any further error processing.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.

6.7 How to Trap Application Exceptions

You can use the `Application_Error()` method of the ASP.NET application file `global.asax` to trap all exceptions that you do not trap at a lower level. You can place the required code inline in `global.asax`, or you can implement a code file for `global.asax`. You can implement application exception management in a code file based on the following example that redirects the web client to a friendly error page:

1. In the Visual Studio web application project, in Visual Studio Solution Explorer, right-click the project or the appropriate directory, then click Add, and then click Class. The Add New Item dialog appears.
2. In the **Add New Item** dialog box, in the Categories tree, select Visual C#.
3. In the **Templates** list, select Class.
4. In the **Name** field, enter `Global.cs`, and then click **Add**.
5. Replace the contents of `Global.cs` with the following, updating the namespace and class name as appropriate:

```
using System;
using System;

namespace Sitecore.Sharedsource.Web.HttpApplication
{
  public class Global : System.Web.HttpApplication
  {
    protected void Application_Error(Object sender, EventArgs e)
    {
      Sitecore.Diagnostics.Log.Error(
        "Application exception in "
        + GetType().ToString() + " processing " + Sitecore.Context.RawUrl,
       System.Web.HttpContext.Current.Server.GetLastError(),
       this);
      Sitecore.Sharedsource.Web.UI.ErrorHelper helper =
        new Sitecore.Sharedsource.Web.UI.ErrorHelper();

      if (helper.ShouldRedirect())
      {
        helper.Redirect();
      }
    }
  }
}
```

6. In Visual Studio Solution Explorer, enable **Show All Files**.
7. In Visual Studio Solution Explorer, right-click `global.asax`, and then click Include In Project.
8. Disable **Show All Files**.
9. Backup the existing `global.asax` file, and then replace its contents with the following, updating the namespace and class name as appropriate:

```
<%@ Application Language="C#"
  Inherits="Sitecore.Sharedsource.Web.HttpApplication.Global" %>
```

The `Application_Error()` method logs the exception, and then uses the error helper described in the section The Error Helper for any further error processing.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2014 Sitecore. All rights reserved.