Sitecore CMS 6.0 or later

Presentation Component Troubleshooting

Problem solving techniques for CMS Administrators and Developers

Table of Contents

Chapter 1

Introduction

This Presentation Component Troubleshooting guide provides problem-solving techniques for CMS administrators and developers to diagnose common issues with presentation components, including performance optimization general troubleshooting procedures.

This document contains the following chapters:

- Chapter 1 — Introduction
- Chapter 2 — Troubleshooting Sitecore
- Chapter 3 — Troubleshooting Presentation Components
- Chapter 4 — Troubleshooting XSL Renderings
- Chapter 5 — Known Issues and Additional Resources

Chapter 2

Troubleshooting Sitecore

This chapter provides general Sitecore troubleshooting information.

This chapter contains the following sections:

- Browser Configuration
- Publishing, Security, Caching, and Languages
- The Sitecore Logs
- Resetting and Restarting IIS and ASP.NET

2.1 Browser Configuration

Always consider browser configuration if you experience an issue with a Sitecore content management user interface.[1]

To attempt to determine if browser configuration could be the cause of an issue, try the same operations from a different client.

[1] For more information about browser configuration and troubleshooting, see http://sdn.sitecore.net/Articles/Administration/Configuring%20IE7.aspx.

2.2 Publishing, Security, Caching, and Languages

Always consider the environment in which code executes, including potential differences between the Master and publishing target databases.[2] Be sure that presentation components access the intended Sitecore database, and that you have published the required information to that database if necessary. If something works in the content management environments but does not work on the published Web site, the most likely issue is that the feature relies on new items or changes in the Master database that Sitecore has not published. Remember to publish changes to data templates and new media referenced by content items before or at the same time as publishing the content items. Remember that publishing will not transfer item and version data due to workflow and publishing restrictions.

If you have published the required items, and there are still differences between the CMS and the published site, the issue could be security. By default, all access the published site occurs as the Anonymous user in the Extranet domain. All access to the CMS occurs as a specific user in the Sitecore domain, which may have different access rights than the Anonymous user in the Extranet domain. You can use the Access Viewer to investigate access rights for the Anonymous user in the Extranet domain.

When properly configured, all types of publishing properly clear any required Sitecore caches. If you use the ASP.NET output cache or custom caches, you are responsible for implementing publishing event handler to invoke a Web service or other custom solution to clear that cache.

Items contain languages, and languages contain versions. An item exist for all languages, and the first version exists for the language in which the user created the item, but the item does not contain field values for any languages for which a version does not exist. Sitecore will not fall back to a default language if no data exists in the requested language. If pages appear empty, check Sitecore's language configuration, and ensure the browser is not requesting the item in a language for which no version data exists.

[2] For more information about publishing and troubleshooting publishing issues, see http://sdn.sitecore.net/Articles/Administration/Sitecore%20Publishing%20Operations.aspx and http://sdn.sitecore.net/End%20User/Site%20Administration/Troubleshooting/Missing%20Content.aspx .

2.3 The Sitecore Logs

Always check for errors and warnings in the Sitecore log entries from around any times that you experience any unexpected behavior. Sitecore generates a log file each day, and additional log files as needed to continue logging when ASP.NET restarts do not immediately release file locks.

The `value` attribute of the `/configuration/sitecore/settings/setting` element in `web.config` with `name LogFolder` specifies the directory containing the Sitecore logs. This setting often references the `$(dataFolder)` variable, which is defined by `value` attribute of the the `/configuration/sitecore/sc:variable` element in `web.config` with `name dataFolder`. With the following settings, Sitecore writes log files in `C:\inetpub\Sitecore\mywebsite\data\logs` directory:

```
<sc.variable name="dataFolder" value="C:\Inetpub\Sitecore\mywebsite\data\" />
...
<setting name="LogFolder" value="$(dataFolder)/logs" />
```

2.3.1 How to Identify Underperforming Requests

Use the Sitecore logs to identify items that contain layout details that reference underperforming presentation components. For each page that takes more than an allowed amount of processing time, Sitecore writes warnings such as the following to its log file:

WARN

Timing threshold exceeded for web page.

Milliseconds

Threshold

Page URL

Sitecore also generates warnings for requests that access more than an allowed number of items, as the number of items accessed by a component affects its performance:

WARN

Item threshold exceeded for web page

Items accessed

Threshold

Page URL

Sitecore also generates warnings for requests that consume more than an allowed amount of memory:

WARN

Memory threshold exceeded for web page

Memory used

Threshold

Page URL

Use these warnings to identify requests that consume inordinate resources. Using the Sitecore browser-based debugger, navigate to the URL specified in the log entry to identify individual components that access large numbers of items, require significant processing time or consume excess memory. For more information about the Sitecore debugger, see the section The Sitecore Browser-Based Debugger.

For components that consume excess memory or processing time, investigate coding issues, and remember to configure output caching. For components that access large numbers of items, consider alternative information architectures.

You can configure warning thresholds by adjusting the properties of the `StopMeasurements` processor in the `httpRequestEnd` pipeline in `web.config`:

```
<processor type="Sitecore.Pipelines.HttpRequest.StopMeasurements, Sitecore.Kernel">
  <TimingThreshold desc="Milliseconds">1000</TimingThreshold>
  <ItemThreshold desc="Item count">1000</ItemThreshold>
  <MemoryThreshold desc="KB">10000</MemoryThreshold>
</processor>
```

Note
It is normal for the first requests after ASP.NET restarts to exceed the allowed thresholds. Ignore threshold warnings immediately after ASP.NET restarts. The `Sitecore started INFO` entry in the Sitecore log indicates an ASP.NET restart.

Note
It is normal that requests for Sitecore user interface components consume more resources than requests for pages on the published site. In general, you can ignore warnings for Sitecore user interface pages, or configure thresholds differently for content management and content delivery environments.

2.4 Resetting and Restarting IIS and ASP.NET

While it is always preferable to investigate an issue to determine the root cause before taking further action, resetting or restarting IIS and ASP.NET can be the fastest technique to return a Sitecore instance to production immediately.

To reset IIS, start a command prompt as a Windows administrator, and execute the following command:

```
iisreset
```

Restarting IIS can resolve more issues than resetting IIS. To restart IIS, start a command prompt as a local Windows administrator and execute the following commands:

```
net stop w3svc
net start w3svc
```

Chapter 3

Troubleshooting Presentation Components

This chapter provides troubleshooting instructions specific to Sitecore presentation components.

This chapter contains the following sections:

- How to Identify Underperforming Presentation Components
- The Sitecore Browser-Based Debugger
- Invalid Placeholder Key or Invalid Presentation Component Nesting Order
- Browser Toolbars

3.1 How to Identify Underperforming Presentation Components

In addition to using the Sitecore logs as described in the section The Sitecore Logs, you can use the presentation component statistics page to identify items containing underperforming presentation components.

The Sitecore installation program configures IIS to deny anonymous access to the `/sitecore/admin` directory containing the presentation component statistics page. To access the presentation component statistics page, you may need to enter Windows credentials in the browser, or allow anonymous access to this directory in IIS. You can enable IP or IIS other restrictions to prevent unauthorized access to the pages in this directory.

To enable anonymous access to the `/sitecore/admin` directory in IIS 6:

1. In the IIS management console, right-click the `/sitecore/admin` directory, and then click Properties
2. Click the Directory Security tab
3. In the Authentication and Access Control section, click Edit.
4. Select Enable anonymous access, and then click OK.

To enable anonymous access to the `/sitecore/admin` directory in IIS 7:

1. In the IIS management console, click the `/sitecore/admin` directory, and then double-click Authentication.
2. Click Anonymous Authentication, and then click Enable.

To access the presentation component statistics page, in the browser, access `/sitecore/admin/stats.aspx` on the Sitecore server. Each row in the report represents a rendering, a placeholder, or a sublayout for a specific logical site (each site has its own caches). The columns provide statistics based on all invocations of the component relative to the time of the last ASP.NET reset or restart. The following table describes values in the presentation components statistics report:

Column	Value
Rendering	Identifies an individual presentation component.
Site	Logical site.
Count	Number of times the layout engine has used the component, whether by invoking it or retrieving output from cache.
From Cache	Number of times the layout engine has used cached output for the component instead of invoking it.
Average Time (ms)	Average processing time per invocation of the component.
Average Items	Average number of items read per invocation of the component.
Max Time	Maximum processing time used by any single invocation of the component.
Max Items	Maximum number of items read by any single invocation of the component.
Total Time	Total processing time for all invocations of the component.
Total Items	Total number of items read by all invocations of the component.
Last Run	Date and time of last invocation of the component.

To identify underperforming components, locate high values for total processing time, maximum number of items accessed, average processing time and maximum processing time. Once you have identified an underperforming component, use the `sc:trace()` XSL extension method to identify the location of the error in an XSL rendering, or use Visual Studio to debug .NET code.[3] For more information about the `sc:trace()` XSL extension method, see the section Write Trace Messages Using the sc:trace() XSL Extension Method.

To identify components without optimal caching configuration, locate components with a high ratio of number of uses (Count) relative to the number of cache hits (From Cache).

Tip
To sort by the various column values, in Internet Explorer, right-click in the table, and then select Export to Microsoft Excel.

[3] For more information about using the Visual Studio debugger, see the Presentation Component Cookbook at http://sdn5.sitecore.net/Reference/Sitecore%206.aspx.

3.2 The Sitecore Browser-Based Debugger

Use the Sitecore browser-based debugger to investigate issues with presentation components. The debugger superimposes debugging user interface elements into pages as you browse the site, including page profiling and tracing features to help identify underperforming components.

The page profile provides a high-level summary of component performance.[4] The page trace provides low-level details of each step in the page assembly process. The page trace also includes error messages not shown on the published site, such as an attempt to bind a presentation component to a placeholder that does not exist.

To access the Sitecore browser-based debugger from within Developer Center, click the Debug menu, and then click Start Debugging. To access the Sitecore debugger from within the Sitecore Desktop, click the Sitecore button, and then click Debug. The debugger opens in a new browser window or tab.

In the Debugger:

- Click Ribbon at the top left corner of the page to show or hide the ribbon.
- In the Browser Views group, choose between Page Editor, Preview, and Debug modes.
- In the Profile group, press the Activate command to control whether the debugger includes the page profile, a high-level performance overview of the age.
- In the Trace group, press the Activate command to control whether the debugger includes the page trace, a detailed summary of the page assembly process.
- In the Rendering group, select the Information checkbox to control whether the debugger includes rendering information, which provides information about each presentation component using a green triangle visible in the debugger.
- Hover over the rendering information icons that appear for each presentation component. In the panel that appears, click on the tabs to see information about the presentation component. The Details tab provides general information about the presentation component, such as the name of the Web control or the path to the XSL file. The Profile tab provides performance information for the presentation component. The Cache Settings tab indicates caching configuration for the presentation component. The Output tab contains the output generated by the individual presentation component.
- Navigate to pages that require analysis, such as pages that do not perform as desired. Investigate rendering information, the page profile, and the page trace to determine the cause of the issue.

Page profile messages include the following information:

- Time: Percentage of entire page rendering time spent on an individual task.
- Action: Description of processing step performed.
- Total: Total task execution time in milliseconds.
- Own: Task execution time in milliseconds, excluding descendant presentation components.
- Items Read: The number of Sitecore items read during a task, which can affect performance.

Important
The Sitecore browser-based debugger accesses the default published site, which uses the Web database by default. To access another database, specify the database name using the `sc_database` URL query string parameter (for example, `sc_database=master`).

[4] For information about optimizing Sitecore performance, see http://sdn.sitecore.net/Articles/Administration/Sitecore%20Performance.aspx.

3.2.1 Disable Rendering Information to Test Caching

When the Sitecore debugger displays rendering information, the layout engine ignores component caching criteria. In this case, the layout engine invokes each presentation component for each page request instead of retrieving output from cache.

To investigate caching configuration, disable rendering information, and investigate the page trace. To disable rendering information, show the ribbon, and in the Rendering group, clear the Information checkbox.

The message `using cache` indicates that the layout engine retrieved output for an XSL rendering from cache.

Note
Messages in the page trace such as `Xslt file loaded from cache` indicate that the system retrieved an XSL transformation object from cache, not the output of that XSL rendering.

3.3 Invalid Placeholder Key or Invalid Presentation Component Nesting Order

If layout details indicate that the layout engine should bind a control to a placeholder, but the layout engine cannot locate a placeholder with that key in the layouts and sublayouts already processed, the layout engine does not create an error visible to the browser. Instead, the layout engine writes warning messages such as the following to the page trace visible in the Sitecore browser-based debugger:

Warning

A rendering was not used on the page

The placeholder was not found

If you locate any of these warning messages in the page trace, confirm that the sublayouts and renderings in layout details for the item specify placeholder keys that exist in the layout and sublayouts specified. Ensure that placeholder settings list controls in component nesting order. For example, a sublayout containing a placeholder should appear before the controls that bind to that placeholder. Ensure that fully qualified placeholder keys reflect the correct placeholder nesting order.

3.4 Browser Toolbars

You can use browser toolbars, such as the Internet Explorer Developer Toolbar, HttpWatch, and Firebug (for Firefox) to diagnose various presentation issues.[5]

Note
Some browser toolbars can interfere with Sitecore solutions, such as by blocking pop-up Windows. You may need to disable or configure browser toolbars to work properly with Sitecore. If you experience an issue that could be related to a browser toolbar, before investigating the issue, recreate the issue from a client that does not have the toolbar installed.

[5] For more information about the Internet Explorer Developer Toolbar, see http://www.microsoft.com/downloads/details.aspx?familyid=e59c3964-672d-4511-bb3e-2d5e1db91038&displaylang=en. For more information about HttpWatch, see http://httpwatch.com. For more information about Firebug, see https://addons.mozilla.org/en-US/firefox/addon/1843.

Chapter 4

Troubleshooting XSL Renderings

This chapter contains procedures to troubleshoot XSL renderings.

This chapter contains the following sections:

- Xsl File Could Not Be Processed
- Extension Object Does Not Contain a Matching Item Method
- Context Element and Specific Items
- Write Trace Messages Using the sc:trace() XSL Extension Method
- Attribute Value Templates
- Investigate Raw XML

4.1 Xsl File Could Not Be Processed

While processing an XSL rendering, if the layout engine encounters a syntax error in the XSL code or an exception a .NET extension throws an exception, the layout engine outputs a warning icon and the following error message:

Xsl file could not be processed

Click on the warning icon and investigate the details of the error. The first few lines contain an error message and possibly the approximate position of the error in source code, which should be useful in diagnosing the problem. The remaining lines contain the .NET stack trace at the point of error.

Note
The Sitecore log also contains the exception details.

4.2 Extension Object Does Not Contain a Matching Item Method

If you neglect to pass the second mandatory parameter to the `sc:item()` XSL extension method, you will receive an error such as the following in the browser:

Extension object http://www.sitecore.net/sc does not contain a matching 'item' method that has 1 parameter(s)

To resolve this issue, pass `$sc_currentitem` or another element representing an item as the second parameter to the `sc:item()` XSL extension method.

For example, the following code does not include the second parameter to the `sc:item()` XSL extension method:

```
        <xsl:variable name="content" select="sc:item('/sitecore/content')" />
```

One possible correction to this code is as follows:

```
        <xsl:variable name="content" select="sc:item('/sitecore/content',$sc_currentitem)" />
```

Note
The specific item passed is arbitrary.

4.3 Context Element and Specific Items

Always consider the context that an XSL construct processes, or explicitly reference specific elements explicitly using the `select` attribute. Otherwise, the XSL construct may generate the wrong output or no output.

For example, a developer may intend to process the items referenced by a Multilist field using the following code:

```
<xsl:for-each select="sc:Split('FieldName',$sc_currentitem)">
  <sc:link>
    <sc:text field="FieldName" />
  </sc:link>
</xsl:for-each>
```

In the `<sc:link>` and `<sc:text>` XSL extension controls in this example, the developer has neglected to set the context item or specify the item to link to and retrieve the field value from using the `select` attribute. The controls default to processing the context element. Due to the for-each, within the `<xsl:for-each>`, the context element is a member of the XML structure returned by the `sc:Split()` XSL extension method, not part of the XML document that represents the Sitecore database.

To correct the issue, specify the item using `select` attributes, or set the context element to that item. To specify the item explicitly using `select` attributes:

```
<xsl:for-each select="sc:Split('FieldName',$sc currentitem)">
  <xsl:variable name="VariableName" select="sc:item(text(),$sc_currentitem)" />
  <sc:link select="$VariableName">
    <sc:text select="$VariableName" field="FieldName" />
  </sc:link>
</xsl:for-each>
```

To set the context item:

```
<xsl:for-each select="sc:Split('FieldName',$sc_currentitem)//value">
  <xsl:for-each select="sc:item(text(),$sc_currentitem)">
    <sc:link><sc:text field="FieldName" /></sc:link>
  </xsl:for-each>
</xsl:for-each>
```

4.4 Write Trace Messages Using the sc:trace() XSL Extension Method

Use the `sc:trace()` XSL extension method to write a message to the page trace visible in the Sitecore browser-based debugger. For example:

```
<xsl:value-of select="sc:trace('Message')" />
```

This use of `<xsl:value-of>` calls the `sc:trace()` XSL extension method, but generates no output.

Note
Excluding custom .NET XSL extensions, the Microsoft Visual Studio debugger does not support stepping through XSL code.

4.5 Attribute Value Templates

When the XSL transformation engine evaluates an element in the `xsl` namespace, such as `<xsl:value-of>`, it automatically evaluates XPath expressions in attribute values. For example, given the following code, the XSL transformation does not output the literal value `sc:path(.)`, but instead evaluates that expression as an XPath statement, in this case a call to the XSL extension method:

```
<xsl:value-of select="sc:path(.)" />
```

The XSL transformation engine does not automatically evaluate XPath expressions in attribute values of other elements. In some cases, this causes the XSL transformation to output XPath expressions instead of the corresponding values, or to generate no output. For example, given the following code, because the `<a>` element is not a member of a recognized namespace such as `xsl` or `sc`, the XSL transformation engine outputs the literal value `<a href="sc:path(.)" />` instead of invoking the XSL extension function `sc:path()` to determine the appropriate value for the `href` attribute:

```
<a href="sc:path(.)" />,
```

The XSL specification defines attribute value templates which cause the XSL transformation engine to evaluate attribute values in unrecognized constructs as XPath statements.[6] Curly braces ("`{}`") indicate attribute value templates and must be used where an XPath statement should be evaluated within an attribute value in non-XSL constructs.

To cause the XSL transformation engine to invoke the `sc:path(.)` XSL extension method to determine the value to output for the `href` attribute of an `<a>` element, wrap the XSL extension method in curly braces:

```
<a href="{sc:path(.)}" />,
```

You can use double-curly braces ("`{{`" and "`}}`") to represent literal individual curly braces where a single curly brace would otherwise cause the XSL transformation engine to interpret a literal attribute value as an attribute value template. For example, given the following code, the XSL transformation engine outputs `<a href="{strangelink" />`:

```
<a href="{{strangelink" />
```

You can add avoid attribute value templates using the `<xsl:element>` and `<xsl:attribute>` XSL elements:

```
<xsl:element name="a">
  <xsl:attribute name="href"><xsl:value-of select="sc:path( . )" /></xsl:attribute>
</xsl:element>
```

Alternatively:

```
<a>
  <xsl:attribute name="href"><xsl:value-of select="sc:path( . )" /></xsl:attribute>
</a>
```

You can also use attribute value templates in XSL extension control attribute values. For example, to use the name of the media item referenced in the field named ImageField in the context item as the `title` attribute of an `<img>` element for that media item:

```
<sc:image title="{sc:item(sc:fld('ImageField',.,'mediaid'),.)/@name}"
  field="ImageField" />
```

[6] For more information about attribute value templates, see section 7.6.2 of http://www.w3.org/TR/xslt#attribute-value-templates.

4.6 Investigate Raw XML

You may wish to review the raw XML available to an XSL rendering.

You can use the `<xsl:copy-of>` XSL element in an XSL rendering to output the contents of an XML structure. For example, you can use `<xsl:copy-of>` to investigate the context item:

```
<xsl:copy-of select="$sc_currentitem" />
```

Note
The `<xsl:copy-of>` element includes all descendants of the selected item. For an item with many decendants, this can consume significant processing resources and generate a great deal of output.

You can use the `<xsl:copy-of>` XSL element to output the contents of any XML structure, such as the data returned by the `sc:Split()` XSL extension method:

```
<xsl:variable name="split" select="sc:Split('FieldName',$sc_currentitem)" />
<xsl:copy-of select="$split" />
```

Note
The following code achieves the same objective without creating a variable:

```
<xsl:copy-of select=" sc:Split('FieldName',$sc_currentitem)" />
```

To view this output, open the rendering in Developer Center, right-click the preview pane, and then choose View Source. Alternatively, open a page that uses the rendering in a browser and view its source.

Alternatively, you can use a layout to return XML in the format available to XSL renderings. For example, create a device named XML, for example triggered by the query string parameter `x=1` and using the icon `Control/32x32/treeview.png`. Create a layout containing the following code:

```
<%@ Page Language="c#" Inherits="System.Web.UI.Page" CodePage="65001"
  ContentType="text/xml" %>

<script runat="server">
protected override void Render(System.Web.UI.HtmlTextWriter writer)
{
  Sitecore.Xml.XPath.ItemNavigator navigator =
    Sitecore.Configuration.Factory.CreateItemNavigator(Sitecore.Context.Item);
  writer.Write(navigator.OuterXml);
}
</script>
```

In layout details for the item to investigate, associate this layout with the XML device, access the item in the browser, and add the `x=1` query string parameter in the browser's address bar.

Chapter 5

Known Issues and Additional Resources

This chapter describes known issues related to troubleshooting presentation components.

This chapter contains the following sections:

- Visual Studio Debugger Becomes Unresponsive
- The Controls Collection Cannot Be Modified Because the Control Contains Code Blocks
- Sitecore.Exceptions.CyclicSublayoutException
- Editing Controls Do Not Appear in the Page Editor
- Additional Troubleshooting Resources

5.1 Visual Studio Debugger Becomes Unresponsive

Visual Studio may become unresponsive when you attempt to attach to an existing process to debug. To resolve this issue:

1. Restart Visual Studio.
2. Open the Web application project.
3. Click the View menu, and then click Solution Explorer.
4. In Visual Studio Solution Explorer, hide all files that are not part of the project.[7]
5. Attach the Visual Studio debugger to the ASP.NET worker process.

[7] For instructions to hide all files in Visual Studio Solution Explorer that are not part of the project, see the Presentation Component Cookbook at http://sdn5.sitecore.net/Reference/Sitecore%206.aspx.

5.2 The Controls Collection Cannot Be Modified Because the Control Contains Code Blocks

You may receive the following message in the browser if a layout or sublayout contains ASP.NET code rendering blocks:

The Controls collection cannot be modified because the control contains code blocks

In general, avoid code rendering blocks.

When necessary, use code rendering blocks only within server controls. For example, the following code contains a code rendering block (the content between `<%=` and `%>`):

```
<script language="javascript">
  alert("<%= myObject.ClientID %>");
</script>
```

One way to correct this code is to place it within a server control, such as a `<div>` element with a value of `server` for the `runat` attribute:

```
<div runat="server">
  <script language="javascript">
    alert("<%=myObject.ClientID%>");
  </script>
</div>
```

5.3 Sitecore.Exceptions.CyclicSublayoutException

Sitecore throws a `Sitecore.Exceptions.CyclicSublayoutException` if layout details or static binding cause a sublayout to bind to a itself, resulting in infinite recursion. The full error message indicates the components involved.

Exception

Sitecore.Exceptions.CyclicSublayoutException

Server Error in / Application.

A sublayout has been recursively embedded within itself

Do not bind a sublayout to itself, either statically, or using layout details.

5.4 Editing Controls Do Not Appear in the Page Editor

The Page Editor may not output editing controls if the layout does not include the ASP.NET root <form> element. To correct this issue, add the <form> element to the layout.

The default <form> element Sitecore includes in new layouts is as follows:

```
<form method="post" runat="server" id="mainform">Insert your controls here.</form>
```

The opening <form> element belongs directly after the opening <body> element in the layout, and the closing </form> element generally belongs immediately before the closing </body> element. This <form> element surrounds all of your other markup and presentation controls.

5.5 Additional Troubleshooting Resources

For additional troubleshooting resources, see:

- The Sitecore Developer Network (http://sdn.sitecore.net).
- The Sitecore Developer Network forums (http://sdn.sitecore.net/Forum.aspx).
- The Internet Explorer configuration and troubleshooting resource on the Sitecore Developer Network (http://sdn.sitecore.net/Articles/Administration/Configuring%20IE7.aspx).
- The product installation documentation on the Sitecore Developer Network (http://sdn.sitecore.net/Products/Sitecore%20V5/Sitecore%20CMS%206/Installation.aspx), including documentation for any optional modules (http://sdn.sitecore.net/Products.aspx).
- The Sitecore Product Installation Troubleshooting guide on the Sitecore Developer Network (http://sdn.sitecore.net/Products/Sitecore%20V5/Sitecore%20CMS%206/Installation%20Troubleshooting.aspx).
- The General Troubleshooting resource on the Sitecore Developer Network (http://sdn.sitecore.net/Articles/Troubleshooting/Troubleshooting%20Sitecore%205,-d-,3.aspx).
- The Troubleshooting FAQ on the Sitecore Developer Network (http://sdn.sitecore.net/FAQ/Troubleshooting.aspx).
- The Site Administrator Troubleshooting resources on the Sitecore Developer Network (http://sdn.sitecore.net/End%20User/Site%20Administration/Troubleshooting.aspx).
- The Performance Troubleshooting resource on the Sitecore Developer Network (http://sdn.sitecore.net/Articles/Administration/Sitecore%20Performance/Miscellaneous/Troubleshooting.aspx).
- The Sitecore contact for your region (http://sitecore.net/contact.aspx).
- The Sitecore Support Portal (http://support.sitecore.net).