

Sitecore CMS 7.0 or later

CMS Performance Tuning Guide

A developer's guide to optimizing the performance of Sitecore CMS

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 © 2001-2013 Sitecore. All rights reserved.

Table of Contents

Chapter 1 Introduction 3
Chapter 2 Tuning Procedures — General 4
2.1 Check SQL Server Index Fragmentation Level 5
2.2 Checking for a SQL Server Maintenance Plan 7
2.3 Cleanup Database Tables 10
2.4 Check Database Cleanup Agents 12
2.5 Software and Server Configuration 14
Chapter 3 Tuning Procedures — Database Properties 16
3.1 Compatibility Level Set To SQL Server 2008 (100) 17
3.2 Auto Close Property Set To False 19
3.3 Auto Shrink Property Set To False 21
3.4 Recovery Model Set to Simple 23
Chapter 4 Tuning Procedures — Sitecore Caches 25
4.1 Setting Initial Cache Values 26
4.2 Tuning Sitecore Caches 29
4.3 Configuring Prefetch Cache Guidelines 32
4.4 Configuring Output (Rendering) Cache Guidelines 34
4.5 Disabling Cache Size Limits 38
Chapter 5 Tuning Procedures — IIS Settings 39
5.1 Enable IIS HTTP keep-alive 40
5.2 IIS Expire Web Content Header 42
5.3 Enable IIS Static Content Compression 44
5.4 Enable IIS Dynamic Content Compression (Optional) 46
Chapter 6 Tuning Procedures — Sitecore Client Optimizations 48
6.1 Check Long Running Validators 49
6.2 Check Excessive Item Versions 51
6.3 Check Excessive Items per Node 53
6.4 Miscellaneous Client Specific Optimizations 55
Chapter 7 Tuning Procedures — Sitecore Miscellaneous Content Delivery Server Optimizations 56
7.1 Disable WebDAV 57
7.2 Disable Performance Counters 59
7.3 Disable Memory Monitor 60
Chapter 8 Tuning Procedures — Sitecore Search 61
8.1 Disable Bucket Debug 62
8.2 More Information 63

The information contained in this document represents the current view of Sitecore Corporation on the issues discussed as of the date of publication and is subject to change at any time without notice. This document and its contents are provided AS IS without warranty of any kind, and should not be interpreted as an offer or commitment on the part of Sitecore, and Sitecore cannot guarantee the accuracy of any information presented. SITECORE MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

The descriptions of other companies' products in this document, if any, are provided only as a convenience to you. Any such references should not be considered an endorsement or support by Sitecore. Sitecore cannot guarantee their accuracy, and the products may change over time. Also, the descriptions are intended as brief highlights to aid understanding, rather than as thorough coverage. For authoritative descriptions of these products, please consult their respective manufacturers.

All trademarks are the property of their respective companies

©2013 Sitecore Corporation. All rights reserved.

Chapter 1

Introduction

This guide is designed as a series of tuning tasks, broken out by relative sections that help insure that the Sitecore implementation is healthy and running at its optimal performance.

The Sitecore CMS Diagnostics Guide is a companion guide that is available to help diagnose performance related issues.

This manual contains the following chapters:

- **Chapter 1 — Introduction**
- **Chapter 2 — Tuning Procedures — General**
- **Chapter 3 — Tuning Procedures — Database Properties**
- **Chapter 4 — Tuning Procedures — Sitecore Caches**
- **Chapter 5 — Tuning Procedures — IIS Settings**
- **Chapter 6 — Tuning Procedures — Sitecore Client Optimizations**
- **Chapter 7 — Tuning Procedures — Sitecore Miscellaneous Content Delivery Server Optimizations**
- **Chapter 8 — Tuning Procedures — Sitecore Search**

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 © 2001-2013 Sitecore. All rights reserved.

Chapter 2

Tuning Procedures — General

Tuning Procedures – General is a series of tasks that are designed to check that the Sitecore implementation is configured to run at its peak performance.

Each task contains introductory information, required skills, symptoms that are likely to occur, procedures for checking the configuration, how to solve, and what results to record.

More specific tuning procedures are explained in the following chapters.

This chapter contains the following sections:

- Check SQL Server Index Fragmentation Level
- Checking for a SQL Server Maintenance Plan
- Cleanup Database Tables
- Check Database Cleanup Agents
- Software and Server Configuration

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 © 2001-2013 Sitecore. All rights reserved.

2.1 Check SQL Server Index Fragmentation Level

As indexes age, insertion and deletion of noncontiguous data can take its toll and cause fragmentation to occur. This can happen in just a few days on a busy CMS database. Minor amounts of fragmentation won't generally hurt performance. But as the percentage of fragmentation increases, performance suffers *dramatically*.

2.1.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

2.1.2 Symptoms

- Dramatic increase in CPU usage.
- Performance degradation on queries.
- Performance degradation on database writes.
- Dropped connections to the database server.
- Slow performance of renderings.
- Slow performance of the Sitecore client tools — Desktop, Content Editor, and so on.

2.1.3 Procedure to Check for Fragmented Indexes

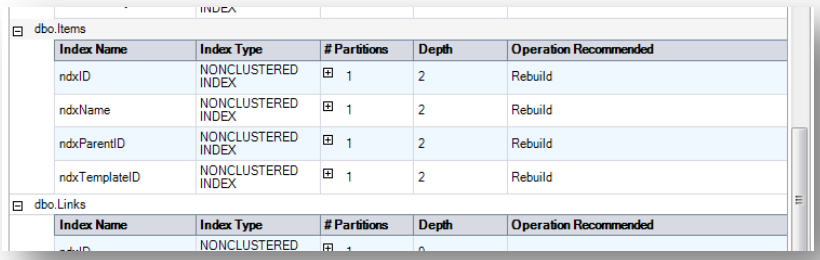
To check for percentage of fragmentation on indexes, run the Index Physical Statistics Standard Report against the CMS databases (Core, Master, Web) as follows:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *Master* database and click **Properties**.
3. Click **Options** page and make sure that the **Compatibility level is set to SQL Server 2008 (100).**
4. Click **OK**.
5. In the **Object Explorer**, right click the *Master* database and click **Reports**, **Standard Reports**, **Index Physical Statistics**.
6. SQL Server Management Studio generates a report showing information about the *Table Names, Index Names, Index Type, Number of Partitions and Operation Recommendations***.**
7. Repeat steps 5 - 6 for the *Core* and *Web* databases to check the level of fragmentation that has occurred on the indexes.

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 © 2001-2013 Sitecore. All rights reserved.

2.1.4 Understanding the Results

The output looks like this:



dbo.Items

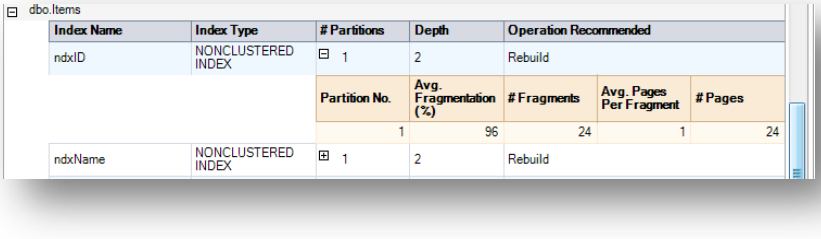
Index Name	Index Type	# Partitions	Depth	Operation Recommended
ndxID	NONCLUSTERED INDEX	1	2	Rebuild
ndxName	NONCLUSTERED INDEX	1	2	Rebuild
ndxParentID	NONCLUSTERED INDEX	1	2	Rebuild
ndxTemplateID	NONCLUSTERED INDEX	1	2	Rebuild

dbo.Links

Index Name	Index Type	# Partitions	Depth	Operation Recommended
	NONCLUSTERED			

One key value that is provided in the report is the Operation Recommended field. Any value of Rebuild is an indication that the index is fragmented.

By expanding the # Partitions field, you can see the % of fragmentation for a given index.



dbo.Items

Index Name	Index Type	# Partitions	Depth	Operation Recommended
ndxID	NONCLUSTERED INDEX	1	2	Rebuild
ndxName	NONCLUSTERED INDEX	1	2	Rebuild

Partition No.	Avg. Fragmentation (%)	# Fragments	Avg. Pages Per Fragment	# Pages
1	96	24	1	24

2.1.5 Sitecore Recommendation

Sitecore recommends keeping index fragmentation below 10%.

2.1.6 How to Solve

In order to defragment the indexes for the CMS databases (Core, Master, Web), execute a defragmentation maintenance plan as follows:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer** expand the **Management / Maintenance Plans** folder.
3. Right click the ***defragment indexes maintenance plan** and click **Execute**.

If such a maintenance plan does not exist, see the section *Checking for a SQL Server Maintenance Plan.*

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 © 2001-2013 Sitecore. All rights reserved.

2.2 Checking for a SQL Server Maintenance Plan

A Maintenance Plan eliminates the need for manual maintenance of the database(s) by running an automated set of tasks on a scheduled basis. This plan performs regular checks and maintenance on the database(s), ensuring that the database(s) in optimal health.

2.2.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.
- A working knowledge of running T-SQL scripts.

2.2.2 Symptoms

- The database is not running to its optimal performance.
- Indexes are becoming fragmented and taken care of.

2.2.3 Procedure to Check for SQL Server Maintenance Plan

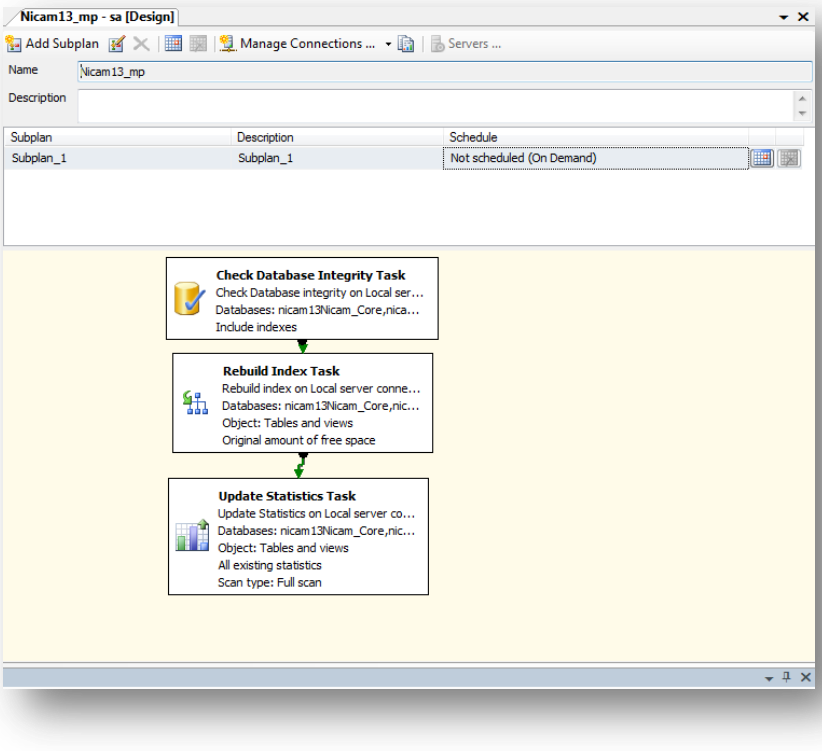
To check for the existence of SQL Server Maintenance Plan, and that it follows Sitecore best practices, do the following:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, expand the **Management / Maintenance Plans** folder.
3. If a maintenance plan exists, double click it to see how it is configured. This is used in the *Findings* section of this task.

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 © 2001-2013 Sitecore. All rights reserved.

2.2.4 Understanding the Results

The output looks like this:



The Maintenance Plan should contain:

- A Check Database Integrity task
- A Rebuild Index Task.
- An Update Statistics task.

2.2.5 Sitecore Recommendation

For the CMS databases Sitecore recommends that a SQL Server Maintenance Plan be in place. The Maintenance Plan contains a Check Database Integrity task, a Rebuild Index task, and an Update Statistics task.

2.2.6 How to Solve

SQL Server Management Studio provides an IDE for the simplification of creating Maintenance Plans. To create a MP for defragmenting the indexes:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer** expand the *Management* folder.
3. Right click the *Maintenance Plans* folder and select **New Maintenance Plan**.
4. Give the MP a meaningful name such as *Defragment CMS Indexes.*

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 © 2001-2013 Sitecore. All rights reserved.

5. From the **Toolbox** drag and drop a **Check Database Integrity Task**, **Rebuild Index Task, Update Statistics Task** and place them vertically in the same order.
6. Connect the tasks together by dragging the arrow from one box to the other so they are connected as: **Check Database Integrity Task -> Rebuild Index Task -> Update Statistics Task**.
7. Right click the **Check Database Integrity Task** and select **Edit**.
8. Select the Connection and CMS databases — Core, Master, Web — and click **OK**.
9. Right click the **Rebuild Index Task** and select **Edit**.
10. Select the Connection and CMS databases — Core, Master, Web — and then select the **Keep index online while reindexing** checkbox — enterprise edition of SQL Server only — and click **OK**.
11. Right click the **Update Statistics Task** and select **Edit**.
12. Select the Connection and CMS databases — Core, Master, Web — and set the Object to Tables and Views, Update All existing statistics, Scan type = Full scan, and click **OK**.
13. Click the calendar icon next to the Schedule (upper right corner) and set the schedule to run weekly.
14. Save.

2.2.7 Notes and Comments

This task shows you how to build a single maintenance plan that rebuilds the indexes for the Core, Master, and Web databases, with a schedule to run the maintenance plan weekly. If you find that some database indexes fragment at a higher rate than others, separate the maintenance plan so that there is a plan for each of the Core, Master, and Web databases. This allows you to create different schedules based on how fast the indexes become fragmented.

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 © 2001-2013 Sitecore. All rights reserved.

2.3 Cleanup Database Tables

There is a series of cleanup agents configured in the *web.config* file to remove artifact data from the *History*, *PublishQueue* and *EventQueue* tables. However, at times these tables become too large which results in timeouts occurring while the agents are trying to do their job.

This task is designed to check the size of the tables, and truncate them if they contain greater than 1000 rows.

2.3.1 Required Skills

- A working knowledge SQL Server Management Studio.
- A working knowledge of running SQL scripts.

2.3.2 Symptoms

- Slow performance during publishing.
- Slow performance during indexing.

2.3.3 Procedure to Clean up the History, PublishQueue, and EventQueue tables

To clean up *the History, PublishQueue, and EventQueue* tables:

1. Launch **SQL Server Management Studio**.
2. Click **New Query**.
3. Run the following query, replacing `/* database name */` with the name of the database that the script runs against**.**

```
USE /* database name */

        /* TRUNCATE History TABLE */
        IF OBJECT_ID('History', 'U') IS NOT NULL
         IF((SELECT COUNT(*) FROM [History]) > 1000)
         BEGIN
           TRUNCATE TABLE [History];
           PRINT 'Truncated the History Table';
         END

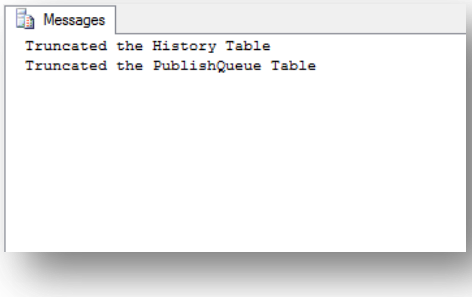
        /* TRUNCATE EventQueue TABLE */
        IF OBJECT_ID('EventQueue', 'U') IS NOT NULL
         if((SELECT COUNT(*) FROM [EventQueue]) > 1000)
         BEGIN
           TRUNCATE TABLE [EventQueue];
           PRINT 'Truncated the EventQueue Table';
         END

        /* TRUNCATE PublishQueue TABLE */
        IF OBJECT_ID('PublishQueue', 'U') IS NOT NULL
         IF((SELECT COUNT(*) FROM [PublishQueue]) > 1000)
           BEGIN
             TRUNCATE TABLE [PublishQueue];
             PRINT 'Truncated the PublishQueue Table';
           END
```

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 © 2001-2013 Sitecore. All rights reserved.

2.3.4 Understanding the Results

The output looks similar to this:



- In the **Messages** window, the tables that were truncated are listed.

2.3.5 Sitecore Recommendation

Sitecore recommends that the number of rows (entries) in the History, PublishQueue, and EventQueue tables be less than 1000. This prevents timeouts from occurring while the cleanup agents run.

2.3.6 How to solve

Run the procedure described earlier.

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 © 2001-2013 Sitecore. All rights reserved.

2.4 Check Database Cleanup Agents

There is a series of cleanup agents configured in the *web.config* file to remove artifact data from the *History*, *PublishQueue* and *EventQueue* tables.

This task is designed to make sure that those tasks are configured and scheduled to run on a periodic basis.

2.4.1 Required Skills

- A working knowledge the Sitecore `web.config` file.

2.4.2 Symptoms

- Slow performance during publishing.
- Slow performance during indexing.

2.4.3 Procedure to Check if the History, PublishQueue, and EventQueue Cleanup Tasks are Configured

The following procedure describes where to look to see if the cleanup tasks are configured to run periodically:

1. Open up the `web.config` file in your favorite editor.
2. Navigate to the `<configuration><sitecore><scheduling></scheduling>` node.

2.4.4 Understanding the Results

The output looks like this — some `<agent>` nodes have been removed for clarity:

```
        <scheduling>
              <!-- Time between checking for scheduled tasks waiting to execute -->
              <frequency>00:05:00</frequency>

              <!-- Agent to clean up history data -->
              <agent type="Sitecore.Tasks.CleanupHistory" method="Run" interval="04:00:00"/>
              <!-- Agent to clean up publishing queue -->
              <agent type="Sitecore.Tasks.CleanupPublishQueue, Sitecore.Kernel" method="Run"
interval="04:00:00">
                <DaysToKeep>30</DaysToKeep>
              </agent>
              <!-- Agent to clean up the event queue -->
              <agent type="Sitecore.Tasks.CleanupEventQueue, Sitecore.Kernel" method="Run"
interval="04:00:00">
                <DaysToKeep>1</DaysToKeep>
              </agent>
```

- Check that the `<scheduling><frequency>` is set to something other than 00:00:00 so that the agents are checked to see if they are ready to be executed.
- Make sure that the interval for *Sitecore.Tasks.CleanupHistory* is greater than **00:00:00**.
- Check that the interval for *Sitecore.Tasks.CleanupPublishQueue* is greater than **00:00:00**.
- Check that the interval for *Sitecore.Tasks.CleanupEventQueue* is greater than **00:00:00**.

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 © 2001-2013 Sitecore. All rights reserved.

2.4.5 Sitecore Recommendation

Sitecore recommends that the *scheduling frequency* be enabled by setting it to a value greater than **00:00:00**, and that cleanup agents for the *History, PublishQueue*, and *EvenQueue* be configured to have an interval greater than **00:00:00**.

The default value for the frequency = **00:05:00** and the default for the intervals = **04:00:00**.

2.4.6 How to Solve

If any of the values for the scheduling frequency, *Sitecore.Tasks.CleanupHistory* interval, *Sitecore.Tasks.CleanupPublishQueue* interval, and/or *Sitecore.Tasks.CleanupEventQueue* is set to **00:00:00**, they should be increased to allow them to run. The default values that are supplied in a non-modified *web.config* file are as follows:

- *Scheduling frequency* = **00:05:00** (5 minutes)
- *Sitecore.Tasks.CleanupHistory* interval = **04:00:00** (4 hours)
- *Sitecore.Tasks.CleanupPublishQueue* interval = **04:00:00** (4 hours)
- *Sitecore.Tasks.CleanupEventQueue* interval = **04:00:00** (4 hours)

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 © 2001-2013 Sitecore. All rights reserved.

2.5 Software and Server Configuration

This check is used to determine if the software (OS, IIS, and SQL Server) and server configuration used meets Sitecore recommended practices for optimal performance. Starting with a minimum baseline, a scoring system is used as a means to record the performance level of the server. The scoring is broken down into sections, with each section having its own value. A score of 1 in any section shows that the section meets the minimum Sitecore recommendation. A score > 1 is able to perform at a higher level. And a score of < 1 is a red flag, indicating that additional resources, or changes, need to be made to bring the server(s) up to at least the Sitecore recommendations.

The scoring system is based on the recommended requirements provided in the installation guide (based on Sitecore version 6.4.x) as listed on the SDN.

Note — Using a Content Delivery Network(CDN)
In the case of high-traffic websites, it is an industry practice to use a CDN to serve certain kinds of assets in order to boost performance. Sitecore recommends the use of a CDN for assets such as media files, documents, CSS files, JavaScript files, etc. where those assets are fairly static and do not require Sitecore security. This allows maximum server resources to be dedicated to delivering dynamic and/or secure content.

For assets that are being served from a CDN, careful consideration should be given to CEP features such as analytics, personalization and engagement automation that could potentially be impacted by serving content from outside of the Sitecore environment.

The use, setup, and configuration of a CDN are beyond the scope of this document.

The following blog entries provide useful information on the subject:

http://blog.image0.com/sitecore/using-sitecore-publishing-pipeline-to-refresh-external-cdn-cache/

http://www.cognifide.com/blogs/sitecore/the-ultimate-approach-to-storing-sitecore-media-items-in-cdn/

2.5.1 Required Skills

- A working knowledge of system infrastructures.

2.5.2 Symptoms

- Poor performance.
- Degradation in performance as traffic increases.

2.5.3 Sitecore Recommendation

Sitecore recommends for all servers:

- All servers should be running in 64-bit mode.
- All servers should be running Windows Server 2008 R2 with the latest security patches.

Sitecore recommends for the Content Authoring Environment (Web Server):

- IIS 7.x

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 © 2001-2013 Sitecore. All rights reserved.

- Quad Core Processor
- 8 GB RAM
- 250 GB

Sitecore recommends for the Content Authoring Environment (SQL Server):

- SQL Server 2008 R2 x64
- Dual Quad Core processors
- 12 GB RAM
- 250 GB disk for data files
- 250 GB disk for log files

Sitecore recommends the following for the Content Delivery Environment (Web Server):

- IIS 7.x
- Dual Quad Core processors
- 8 GB RAM
- 250 GB disk

Sitecore recommends the following for the Content Deliver Environment (SQL Server):

- SQL Server 2008 R2 x64
- Dual Quad Core processors
- 12 GB RAM
- 250 GB disk for data files
- 250 GB disk for log files

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 © 2001-2013 Sitecore. All rights reserved.

Chapter 3

Tuning Procedures — Database Properties

Tuning Procedures – Database Properties contains a series of tasks designed to check the configuration of SQL Server database properties. Proper configuration of these properties help the Sitecore implementation run at its peak performance.

This chapter contains the following sections:

- Compatibility Level Set To SQL Server 2008 (100)
- Auto Close Property Set To False
- Auto Shrink Property Set To False
- Recovery Model Set to Simple

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 © 2001-2013 Sitecore. All rights reserved.

3.1 Compatibility Level Set To SQL Server 2008 (100)

Compatibility Level effects SQL syntax and query parsing, and should have no impact of performance. Setting the Compatibility Level to a value of SQL Server 2008(100) takes advantage of new T-SQL features, which are used in many of the scripts / commands used in the CMS Performance Tuning Guide.

3.1.1 Required Skills

- A working knowledge of SQL Server Management Studio.

3.1.2 Symptoms

- Inability to run scripts to improve performance.

3.1.3 Procedure to Check the Database Compatibility Level

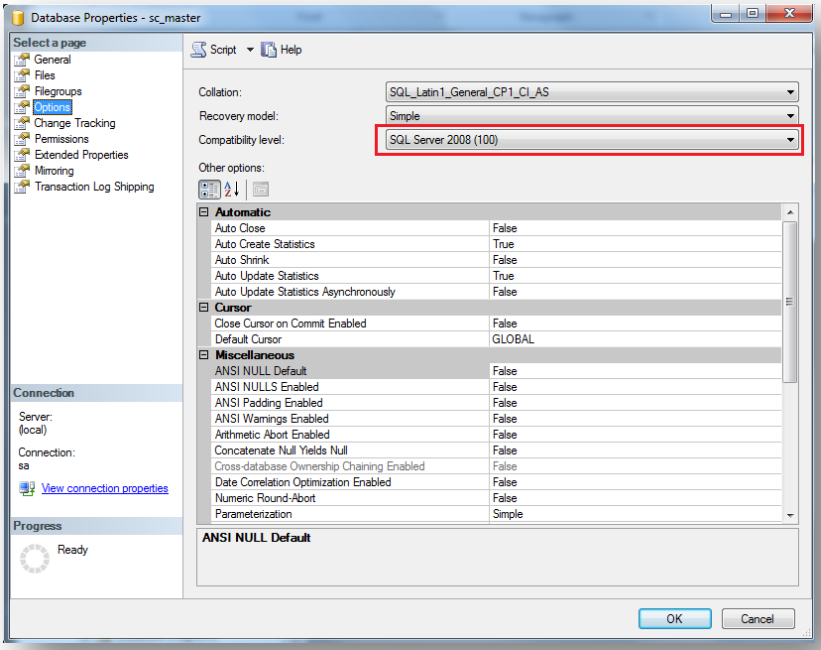
To check for the value of the Database Compatibility Level:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *CMS Master* database and click **Properties**.
3. Click the **Options** page and look at the *Compatibility Level* property.
4. Click **OK**.
5. Repeat steps 1 - 4 for the *CMS Core and Web* 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 © 2001-2013 Sitecore. All rights reserved.

3.1.4 Understanding the Results

The output looks like this:



3.1.5 Sitecore Recommendation

Sitecore recommends that you set the *Compatibility Level* property to **SQL Server 2008(100).**

3.1.6 How to solve

In order to set the *Compatibility Level* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *CMS Master* database and click **Properties**.
3. Click **Options** page and make sure that the **Compatibility Level** is set to **SQL Server 2008(100)**.
4. Click **OK**.
5. Repeat steps 1 - 4 for the *CMS Core* and *Web* 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 © 2001-2013 Sitecore. All rights reserved.

3.2 Auto Close Property Set To False

When SQL Server opens a database, resources are allocated to maintain that state. Memory for locks, buffers, security tokens, etc. are all assigned. These operations take time. The Auto Close property dictates how these resources are handled. If it is set to 'true' or 'ON', then when the last connection is closed these resources are deallocated. This may seem like a good thing, but if a new connection comes in within a short period of time (1/10th of second or quicker), then all of those resources need to be spun up again. Setting the *Auto Close* property to *false* or *OFF* prevents this from happening.

3.2.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.2.2 Symptoms

- Longer times required connecting to the database.

3.2.3 Procedure to check for Auto Close Property value

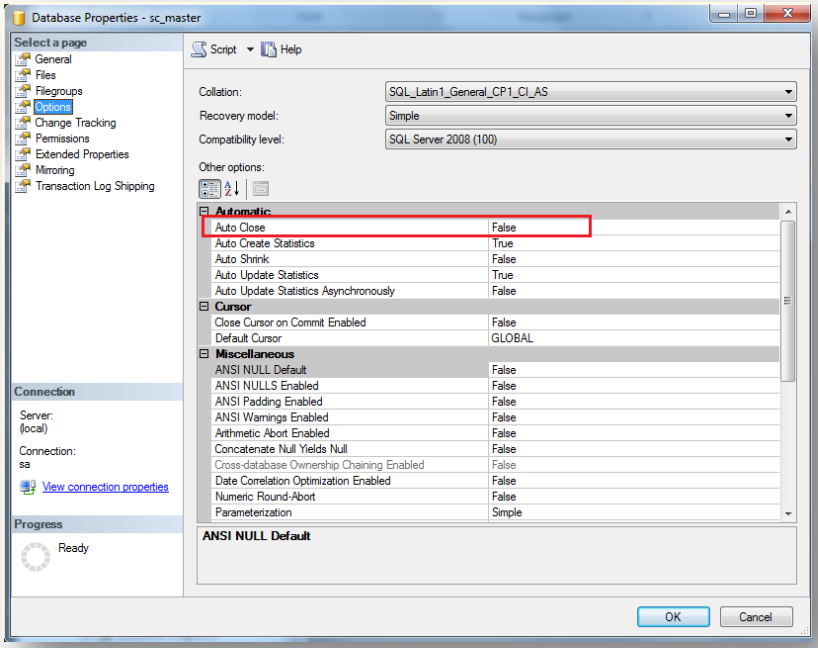
To check for the value of the *Auto Close* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *CMS Master* database, and click **Properties**.
3. Click **Options** page and look at the *Auto Close* property.
4. Click **OK**.
5. Repeat steps 1 - 4 for the *CMS Core* and *Web* 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 © 2001-2013 Sitecore. All rights reserved.

3.2.4 Understanding the Results

The output looks like this:



3.2.5 Sitecore Recommendation

Sitecore recommends that the *Auto Close* property be set to *False*.

3.2.6 How to Solve

To set the *Auto Close* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *CMS Master* database, and click **Properties**.
3. Click **Options** page and make sure that the *Auto Close* property is set to *False*.
4. Click **OK**.
5. Repeat steps 1 - 4 for the *CMS Core* and *Web* 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 © 2001-2013 Sitecore. All rights reserved.

3.3 Auto Shrink Property Set To False

The *Auto Shrink* property has the downside of a) it uses a lot of resources when it's called, and b) you have no control over when it is being called. If you combine *Auto Shrink* with *Auto Growth*, you can get into a spiral of constantly growing and shrink the database, taking valuable resources away from other database tasks as well as causing fragmentation issues. If a database or file requires a SHRINK command, it should be done so via a script, command or scheduled Maintenance Plan. Setting the *Auto Shrink* property to *false* or *OFF* disables this feature.

3.3.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.3.2 Symptoms

- Performance degradation.

3.3.3 Procedure to Check the Auto Shrink Property Value

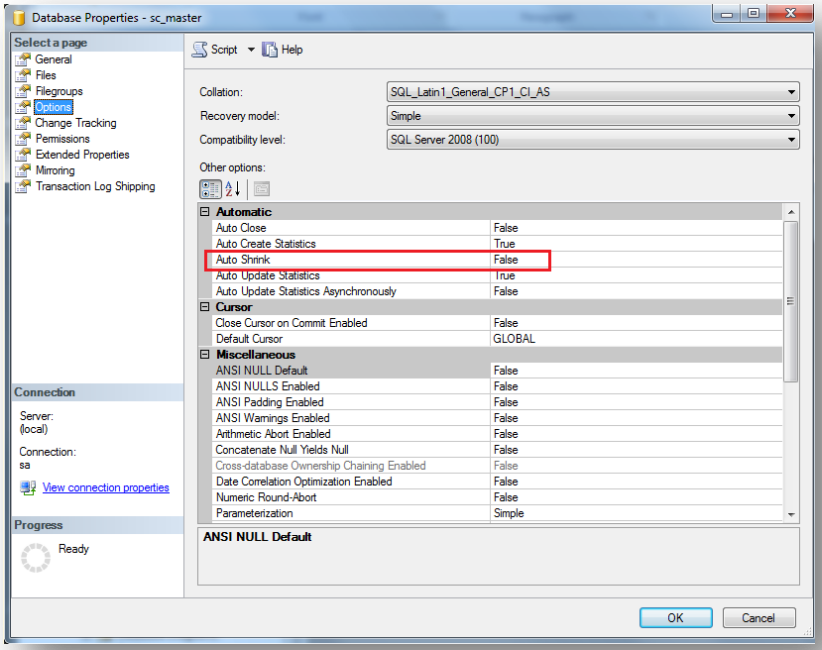
To check for the value of the *Auto Shrink* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *CMS Master* database, and click **Properties**.
3. Click **Options** page and look at the *Auto Shrink* property.
4. Click **OK**.
5. Repeat steps 1 - 4 for the *CMS Core* and *Web* 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 © 2001-2013 Sitecore. All rights reserved.

3.3.4 Understanding the Results

The output looks like this:



3.3.5 Sitecore Recommendation

Sitecore recommends that the *Auto Shrink* property be set to *False*.

3.3.6 How to Solve

To set the *Auto Shrink* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *CMS Master* database, and click **Properties**.
3. Click **Options** page and make sure that the *Auto Shrink* property is set to *False*.
4. Click **OK**.
5. Repeat steps 1 - 4 for the *CMS Core* and *Web* 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 © 2001-2013 Sitecore. All rights reserved.

3.4 Recovery Model Set to Simple

In Simple Recovery Model SQL Server logs minimal amount of information in the transaction log. SQL Server basically truncates the transaction log whenever the transaction log becomes 70 percent full or the active portion of the transaction log exceeds the size that SQL Server could recover in the amount of time, which is specified in the Recovery Interval server level configuration.

Setting Recovery Model to Simple has the lowest amount of overhead over Full and Bulk-logged, which is crucial to the performance requirements needed for the CMS databases.

3.4.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.4.2 Symptoms

- Performance degradation during recovery intervals.

3.4.3 Procedure to Check the Recovery Model Property Value

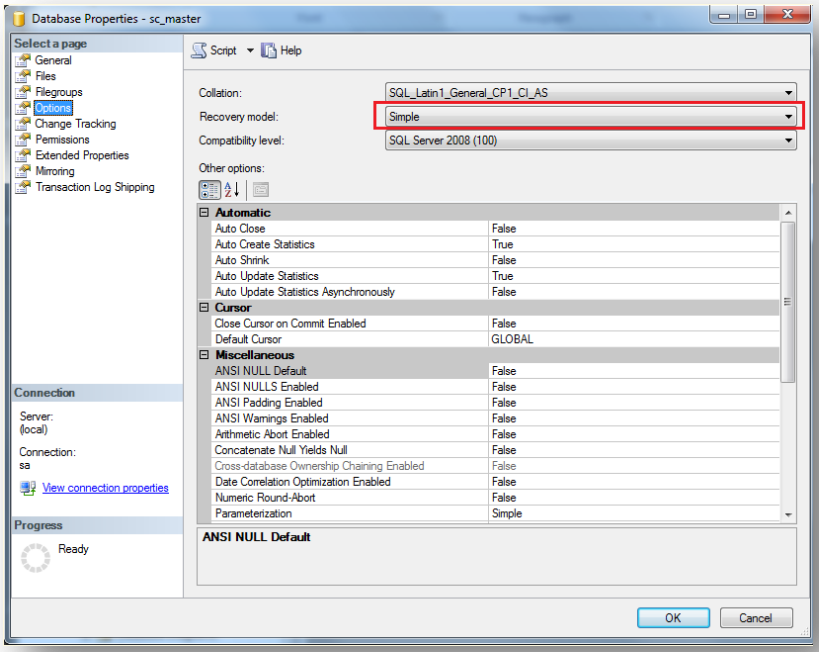
To check the value of the *Recovery Model* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *CMS Master* database, and click **Properties**.
3. Click **Options** page and look at the *Recovery Model* property.
4. Click **OK**.
5. Repeat steps 1 - 4 for the *CMS Core* and *Web* 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 © 2001-2013 Sitecore. All rights reserved.

3.4.4 Understanding theResults

The output looks like this:



3.4.5 Sitecore Recommendation

Sitecore recommends that you set the *Recovery Model* property to *Simple*.

3.4.6 How to Solve

To set the *Recovery Model* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *CMS Master* database, and click **Properties**.
3. Click **Options** page and make sure that the *Recovery Model* property is set to *Simple*.
4. Click **OK**.
5. Repeat steps 1 - 4 for the *CMS Core* and *Web* 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 © 2001-2013 Sitecore. All rights reserved.

Chapter 4

Tuning Procedures — Sitecore Caches

Investigation and configuration of the Sitecore Caches is broken down into multiple tasks. This way each task is more focused and simplified. The focus is on configuration and tuning of the Sitecore Database Caches (prefetch, data, and item caches.

For configuration of the output rendering caching properties, the customer should be made aware of both the *Sitecore Cache Configuration Reference* and the *Sitecore Presentation Component Reference* as to how properly enable and the properties to expire these caches.

Please note that configuration and tuning of the Sitecore Caches should only be performed on a test or develop environment, and never in production since changes to the caches flushes them. This could lead to poor user experience in the production environment.

Once the caches have been tuned, the changes made to the configuration files can be transferred to the production environment during an off peak time.

If you are running Sitecore 6.4.1 Update-4 (rev. 110928) or greater in a 64-bit environment you can disable cache size limits, see the section *Disabling Cache Size Limits*.

This chapter contains the following sections:

- Setting Initial Cache Values
- Tuning Sitecore Caches
- Configuring Prefetch Cache Guidelines
- Configuring Output (Rendering) Cache Guidelines
- Disabling Cache Size Limits

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 © 2001-2013 Sitecore. All rights reserved.

4.1 Setting Initial Cache Values

This task sets the initial values for the Sitecore Database and html output Caches, depending on the environment being addressed. These values are used during the tuning phase of the caches.

Warning
Changing of cache values should only be performed on a test or development environment. Once the caches have been tuned, the new cache values can be transferred to the production environment with the understanding that the caches are flushed once the changes have been saved.

4.1.1 Required Skills

- A working knowledge the Sitecore configuration files.

4.1.2 Symptoms

- Limited amount of memory to caches
- Frequent eviction of caches due to limited memory
- Slow page rendering performance

4.1.3 Initial Values

The following table indicates a good starting point for setting the database and html output caches. These values are based on a) environment, and b) that you are running in 64 bit mode. For a system that is running in 32 bit mode, values should be halved. Note, we strongly recommended that you do not run in 32 bit mode due to memory availability restrictions.

Environment / Target	Cache	Value
Content Delivery Only		
Web	Prefetch	200 MB
Web	Data	200 MB
Web	Item	200 MB
Output (per site)	html	100 MB
CMS Only		
Master	Prefetch	200 MB
Master	Data	200MB
Master	Item	200 MB
CMS and Content Delivery on same server (master and web 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 © 2001-2013 Sitecore. All rights reserved.

Environment / Target	Cache	Value
Master	Prefetch	200 MB
Master	Data	200 MB
Master	Item	200 MB
Web	Prefetch	150 MB
Web	Data	150 MB
Web	Item	150 MB
Output (per site)	html	100 MB
CMS and Content Deliver on same server (Live mode)		
Master	Prefetch	300 MB
Master	Data	300 MB
Master	Item	300 MB
Output (per site)	html	100 MB

4.1.4 Procedure to Set Data and Item Caches to Initial Values

To set the data and item caches to:

1. Open up the `web.config` file in an editor — located in the web root directory.
2. Navigate to `<configuration><sitecore><databases><database id="x"><cacheSizes>` — where "x" = the database listed in the initial values table above.

```
<cacheSizes hint="setting">
        <data>20MB</data>
        <items>10MB</items>
        <paths>500KB</paths>
        <standardValues>500KB</standardValues>
        </cacheSizes>
```

3. Set the initial values for the data and item database caches.
4. Repeat for each database, based on the environment you are running.

4.1.5 Procedure to Set the Prefetch Caches to Initial Values

To set the prefetch caches to initial values:

1. The prefetch cache configuration is located in the `App_Config\Prefetch` directory. Each database — Core, Master, and Web — have their own configuration file. Based on the initial values table above, edit the appropriate file for the database you are working with.
2. Open the prefetch configuration file.

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 © 2001-2013 Sitecore. All rights reserved.

3. Navigate to `<configuration><cacheSize>`

```
<configuration>
  <cacheSize>20MB</cacheSize>
```

4. Set the initial value for the prefetch cache.
5. Repeat for each database, based on the environment you are running.

4.1.6 Procedure to Set the Html Output Caches to Initial Values

To set the data and item caches to:

1. Open up the `web.config` file in an editor — located in the web root directory.
2. Navigate to `<configuration><sitecore><sites><site name="website">`

```
<site name="website" virtualFolder="/" physicalFolder="/" rootPath="/sitecore/content"
startItem="/home" database="web" domain="extranet" allowDebug="true" cacheHtml="true"
htmlCacheSize="100MB" registryCacheSize="0" viewStateCacheSize="0" xslCacheSize="5MB"
filteredItemsCacheSize="2MB" enablePreview="true" enableWebEdit="true" enableDebugger="true"
disableClientData="false"/>
```

3. Set the initial values for the html output cache.
4. Repeat for each website, based on the environment you are running.

4.1.7 Sitecore Recommendation

Sitecore recommends that the prefetch, data, item, and html output caches be set to initial values prior to performing cache tuning.

4.1.8 How to Solve

Follow the procedures described earlier about how to set the initial values for the prefetch, data, item, and html output caches based on your environment.

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 © 2001-2013 Sitecore. All rights reserved.

4.2 Tuning Sitecore Caches

This task describe the procedures used to tune the Sitecore prefetch, data, and item caches, using a load generator and the Sitecore cache.aspx page to monitor the size and eviction of the various caches. For information about configuration of the output (HTML) and rendering cache, see the *Sitecore Cache Configuration Reference* and the *Sitecore Presentation Component Reference* manuals.

Cache tuning is an ongoing process that needs to be revisited from time to time. As the website grows with additional content / items, the sizes of the caches may need to be modified to keep the site running at its optimal performance levels.

4.2.1 Required Skills

- A working knowledge of the cache.aspx page.
- A working knowledge of Sitecore configuration files.
- A working knowledge of a load generation tool such as WCAT or Web Performance Load Tester.

4.2.2 Symptoms

- Poor solution performance.
- Excessive server load.
- Decreased system capacity.

4.2.3 Procedure

Tuning the Sitecore database caches is an iterative process that involves running a generated load against your test or development site, checking to see how the size of the cache compares to its max size (or initial values) as well as the eviction rate, modifying the cache values, and re-checking.

4.2.4 Prerequisites

The Sitecore database caches need to be set to initial values prior to tuning. Refer to: *Setting Initial Cache Values*

A load generation tool and script is needed to exercise the caches. For more information about programs to generate web load, see the *Using WCAT to Generate Load* or *Using Web Performance Load Tester*.

(Sitecore Recommended) The OS System Type is running in 64-bit mode.

4.2.5 Procedure for tuning the Sitecore database caches

1. Run a load generator that hits all items in all languages. The time that the load generator runs should be long enough to go through the entire tuning process, so that it does not require starting and stopping during the process.
2. While the load generator is running, navigate to /sitecore/admin/cache.aspx page. The caches of interest, depending on environment, are the Master[data], Master[items], Web[data], Web[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 © 2001-2013 Sitecore. All rights reserved.

SqlDataProvider- Prefetch data(master), and SqlDataProvider - Prefetch data(web).

Name	Count	Size	Delta	MaxSize
master[data]	6732	20MB	20MB	50MB
master[items]	9816	47MB	47MB	75MB
web[data]	0	0	0	20MB
web[items]	0	0	0	10MB
SqlDataProvider - Prefetch data(master)	7302	97MB	97MB	100MB

3. In order to tune the caches you must hit the refresh button on the page (not the browser refresh) often to see how the caches are reacting to the load. What you want to look at is how the Size of the cache compares to the MaxSize, as well as the fluctuations that occur with the Delta value. Following this, here are some guidelines for adjusting the cache sizes (please note that the maximum amount of memory that you can assign to caches is dependent on the amount of available memory that is in the system. The more memory that you have the better [see hardware recommendation task]):
4. If the Delta for the cache fluctuates constantly, or if the size of the cache is consistently > 80% of the MaxSize, increase the size of the caches by 50%.
5. If the size of the cache remains < 50% of the MaxSize, decrease the size of the cache by 25% to reduce memory consumption.
6. Repeat step 3 until the caches are stable. Ideally you would like to see the cache sizes be between 70% and 80% without constant fluctuations to the Delta.

4.2.6 Understanding the Results

[Include Screen shots of cache.aspx page]

4.2.7 Sitecore Recommendation

Sitecore recommends that the Sitecore database cache size(s) is between 70% and 80% of their MaxSize, without seeing constant fluctuations to the Delta value.

4.2.8 How to Solve

Follow the procedure for tuning the Sitecore database caches described earlier.

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 © 2001-2013 Sitecore. All rights reserved.

Report Findings

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 © 2001-2013 Sitecore. All rights reserved.

4.3 Configuring Prefetch Cache Guidelines

Configuring Prefetch Cache Guidelines is not a task, but more of a primer on what the prefetch caches are, as well as recommended practices on how to configure and utilize them.

It is highly recommended that the Sitecore Cache Configuration Reference be read to gain a full understanding of Sitecore Caching. Refer to:
http://sdn.sitecore.net/upload/sitecore6/sc62keywords/cache_configuration_reference_us.pdf

4.3.1 Understanding the Prefetch Cache

Prefetch caches are populated at application initialization, based on the information provided in the prefecth configuration files. This results in a smoother user experience after an application restart. However, excessive use of the prefetch cache can increase the time required for the application to restart, giving a negative user experience.

To understand how the prefetch caches are utilized once the application is up and running, a view into how Sitecore caching works is required:

When a database item cache does not contain an entry for an item, Sitecore retrieves the corresponding entry from the database data cache, converts it to a different type, and stores that converted data as an entry in the database item cache. When the database data cache does not contain an entry for an item, Sitecore retrieves the corresponding entry from the database prefetch cache, converts it to a different type, and stores that converted data as an entry in the database data cache. When an entry does not exist for an item in a database prefetch cache, Sitecore retrieves that item from the data provider for that database, converts it to a different type, and stores that converted data as an entry in the database prefetch cache.

What this means is that the prefetch caches are not only populated at initialization, but during the life of the application. This understanding is required when thinking about setting the size of the prefetch caches.

4.3.2 Understanding Prefetch Cache Configuration Files

The prefetch cache configuration files can be found in the `/App_Config/Prefetch` directory. Each database has its own prefetch cache configuration file (`Core.config, Master.config, Web.config`) which are merged with the `/App_Config/Prefetch/Common.config` file.

Note that items, templates, and so on, are identified in the configuration files by their GUIDs.

Configuration Elements

- `<configuration>` — This is the root node of the configuration file.
- `<cacheSize>` — This controls the maximum size of the prefetch cache. For example: `<cacheSize>100MB</cacheSize>`.
- `<item>` — This element tells Sitecore the specific item to load into the prefetch cache at application initialization. For example: `<item desc="home">{110D559F-DEA5-42EA-9C1C-8A5DF7E70EF9}</item>`.
- `<template>` — This element tells Sitecore to cache all items based on a given template at application initialization For example: `<template desc="layout">{3A45A723-64EE-4919-9D41-02FD40FD1466}</template>`.

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 © 2001-2013 Sitecore. All rights reserved.

- `<children>` — This element tells Sitecore to cache all of the children of the specified item. (For example: `<children desc="main items">{110D559F-DEA5-42EA-9C1C-8A5DF7E70EF9}</children>`.
- `<childLimit>` – This defines the number of children to cache. If an item has more children than this number, Sitecore does not cache the children. For example: `<childLimit>100</childLimit>`.

4.3.3 Prefetch Cache Recommended Practices

This is a list of recommended practices that provide guidance when configuring the Sitecore Prefetch caches:

- To take advantage of Sitecore Prefetch caching, you must configure it. The information that is in the default prefetch caching files is, in many cases, is not accurate. For example, the `/App_Config/Prefetch/Master.config` file as several entries for html related controls. These controls are now stored in the Core database.
- When configuring the size of the prefetch caches, remember that the prefetch caches are populated throughout the lifecycle of the application, and not just at application initialization.
- Monitor the growth, size and eviction of the prefetch caches the same as you would the item and data caches. Refer to the *Tuning Sitecore Caches* task above for more information.
- It is recommended that the `<setting name="ContentEditor.RenderCollapsedSections" value="false" />` be added to the `/web.config` file. This is a hidden setting that is set to "true" by default. Keeping this setting as "true" (default) makes Sitecore load children of collapsed items into memory (cache) due to the Content Editor implementation + load children of their children due to the prefetching. This makes Sitecore load 2 extra levels of items, when for example: showing the Home item, and slowing down the operation.

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 © 2001-2013 Sitecore. All rights reserved.

4.4 Configuring Output (Rendering) Cache Guidelines

Configuring HTML Output (Rendering) Cache Guidelines is not a task, but more of a primer on what the Sitecore output caching is, as well as recommended practices on how to configure and utilize them.

We recommended that you read chapter 4, *Output Caching*, of the *Sitecore Present Component Reference* to gain a full understanding of Sitecore Output Caching. Refer to: http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Reference.aspx

4.4.1 Understanding the Output (Rendering) Cache

Utilizing the Sitecore output caching can greatly improve the performance of a website. By taking the results of a rendering operation and displaying those results from memory, rather than executing the rendering code, the process is much faster.

The Sitecore output caching is associated with each managed Web site.

Sitecore output caching allows the developer to pick and choose which rendering components to cache, as well as VaryBy parameters that define rules on when those caches are to be evicted. This means that common page components such as headers and footers can be cached, while dynamic components such as news feeds can remain dynamic, all within the same resulting webpage.

The default behavior of the layout engine is to not cache presentation components. To take advantage of the Sitecore output caching, presentation components must be configured properly. This section provides guidance and recommended practices on how to do so.

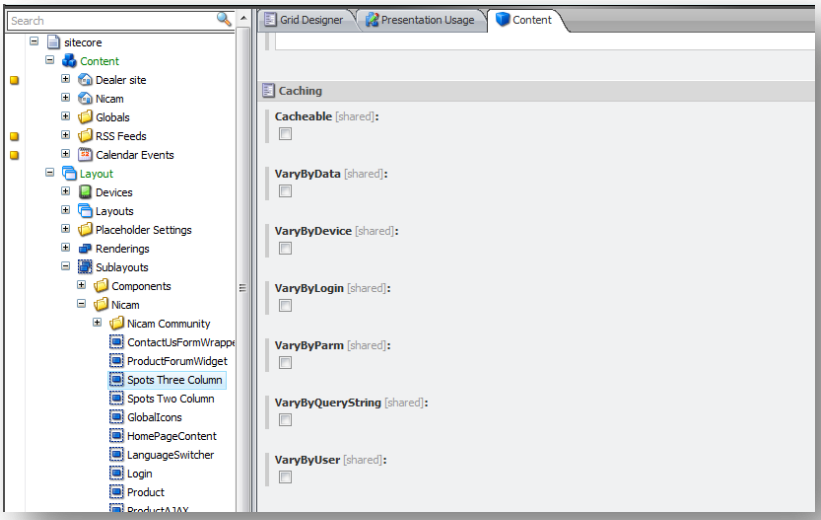
The Sitecore output caching should not be confused with the ASP.NET page and fragment caching (implemented with the `OutputCache` directive in Web forms and Web user controls). Developers should not use Sitecore output caching with ASP.NET page and fragment caching without an understanding of how to properly clear the ASP.NET caches after an operation such as publishing.

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 © 2001-2013 Sitecore. All rights reserved.

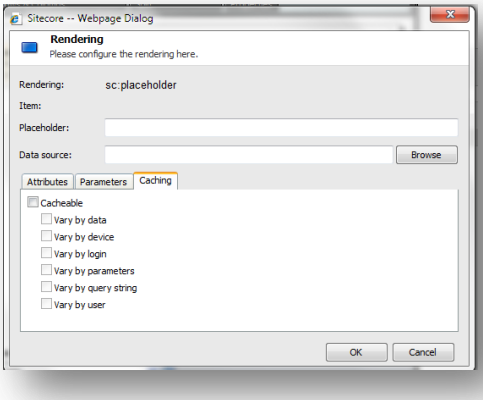
4.4.2 Where to Configure Cache Settings and Which Cache Settings Apply

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

- In the **Caching** section of the sublayout and rendering definition item.



- In the properties of the presentation component, when you statically bind it to a layout or sublayout.

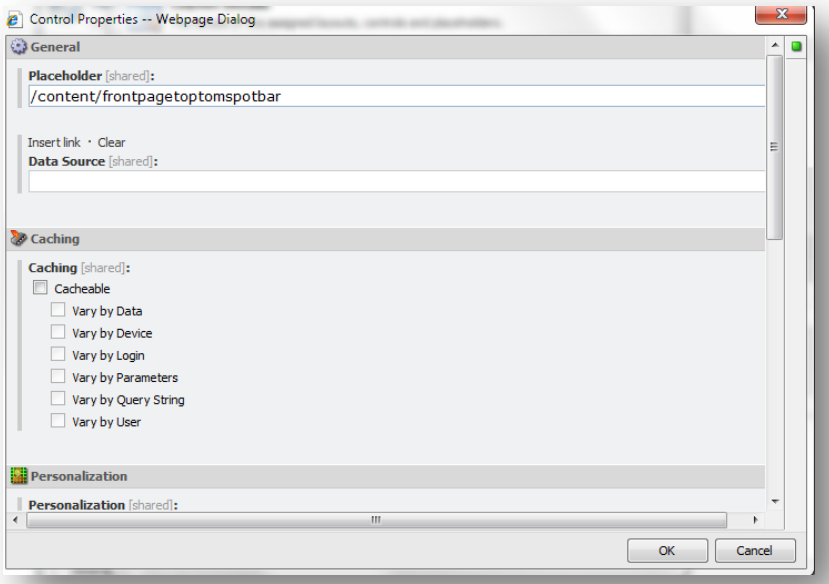


- In declarative code:

```
<sc:sublayout runat="server" renderingid="D366A65-54FF-49B5-A57F-
2EBB9F426433%7d" placeholder="content" cacheable="true" varybydata="true"
path="/layouts/Starter Kit/Sublayouts/Header Fixed.ascx"
id="HeaderFixed"></sc:sublayout>
```

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 © 2001-2013 Sitecore. All rights reserved.

- In the **Caching** section of the **Control Properties** dialog box, when you bind a presentation component to a placeholder in a layout details.



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

- When a developer statically binds a rendering to a layout or sublayout, Sitecore copies caching properties from the rendering definition item to the control (a static reference to the presentation component). Note, the caching properties must exist in the definition item for this to occur. If the layout or sublayout is statically bound prior to the caching properties being set in the rendering definition item, they are not copied.
- When layout details do not specify caching criteria for presentation components dynamically bound to placeholders.

When you dynamically bind a rendering to a placeholder using layout details, cache settings explicitly defined in layout details override cache settings defined in the rendering definition item. Cache settings defined in the definition item apply only when no caching settings exist in the **Caching** section in the **Control Properties** dialog box.

4.4.3 Output Caching Properties

Presentation components that are cacheable have the following properties:

- **Cacheable** — The **Cacheable** property defines whether or not the presentation component should be cached, regardless of what the *VaryBy* properties are set to. If false, the presentation component is invoked every time it is requested. If true, the first request the presentation component is invoked, and then it is retrieved from cache for all subsequent requests. If true and one or more *VaryBy* properties are true, then those *VaryBy* properties control when the presentation component is to be invoked or taken from cache.
- VaryBy properties — The *VaryBy* properties provide control over when a presentation component is to be invoked or taken from cache. The VarBy properties only take effect if Cacheable is set to 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 © 2001-2013 Sitecore. All rights reserved.

- **VaryByData** — Set to true for components that generate different output when used with different data sources.
- **VaryByDevice** — Set to true for components that generate different output when used with different devices.
- **VaryByLogin** — Set to true for components that generate different output for an authenticated vs. an unauthenticated user. Note, the layout engine treats all anonymous users as a single authenticated user.
- **VaryByParm** — Set to true for components that generate a different output when different rendering parameters are passed in.
- **VaryByQueryString** — Set to true for components that generate a different output when different query string parameters are passed in.
- **VaryByUser** — Set to true for components that generate different output for different users. Note, to avoid excessive memory usage only use *VaryByUser* on solutions with a relatively small number of users.

Note
If you statically place renderings in layouts and sublayouts using Visual Studio, or another form of editor, you can manually set the caching properties. For example: `<sc:sublayout runat="server" renderingid="D366A65-54FF-49B5-A57F-2EBB9F426433%7d" placeholder="content"` cacheable="true" varybydata="true" `path="/layouts/Starter Kit/Sublayouts/Header Fixed.ascx" id="HeaderFixed"></sc:sublayout>`

4.4.4 Output Caching Recommended Practices

This is a list of recommended practices that provides guidance when configuring the Sitecore Output caching:

- Sitecore output caching can provide large gains in rendering performance by reducing CPU utilization required to invoke presentation components. To gain the most out of using output caching, insure that good coding practices are followed prior to making presentation components cacheable.
- Make sure that the `htmlCacheSize` property is large enough to cache all renderings. For more information, see the section *Setting Initial Cache Values*.
- Caching of a container (layout, placeholder, or sublayout) that contains child controls result in taking the entire rendering result of the container from cache on subsequent requests. This includes any child controls that exists within the container.
- Use the stats.aspx page to monitor caching activity and rendering times for the presentation components.
- Information on creating custom caching criteria can be found on the Sitecore Blog at: http://www.sitecore.net/Community/Technical-Blogs/John-West-Sitecore-Blog/Posts/2011/05/Custom-Caching-Criteria-with-the-Sitecore-ASPNET-CMS.aspx

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 © 2001-2013 Sitecore. All rights reserved.

4.5 Disabling Cache Size Limits

This task disables the cache size limits for systems using Sitecore 6.5.1 Update-4 or greater. This is to be used on 64-bit systems with ample amounts of memory to allow caching to utilize available memory, without restricting memory usage from other parts of the system

4.5.1 Required Skills

- A working knowledge the Sitecore configuration files.

4.5.2 Symptoms

- Limited amount of memory to caches
- Frequent eviction of caches due to limited memory
- Slow page rendering performance

4.5.3 Procedure to Disable Cache Size Limits

To disable the cache size restrictions:

1. Open up the `web.config` file in an editor (located in the web root directory).
2. Navigate to `<setting name="Caching.DisableCacheSizeLimits" value="false" />` and set the value="true".

```
<setting name="Caching.DisableCacheSizeLimits" value="true" />
```

3. Save your changes.

4.5.4 Sitecore Recommendation

Sitecore recommends that the cache size limits are disabled on 64-bit systems with ample memory.

4.5.5 How to Solve

Follow the procedures described to set the `Caching.DisableCacheSizeLimits` setting value="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 © 2001-2013 Sitecore. All rights reserved.

Chapter 5

Tuning Procedures — IIS Settings

Tuning Procedures – IIS Settings contains a series of tasks designed to check the configuration of the IIS web server. Proper configuration of IIS helps the Sitecore implementation run at its peak performance.

This chapter contains the following sections:

- Enable IIS HTTP keep-alive
- IIS Expire Web Content Header
- Enable IIS Static Content Compression
- Enable IIS Dynamic Content Compression (Optional)

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 © 2001-2013 Sitecore. All rights reserved.

5.1 Enable IIS HTTP keep-alive

Enabling the HTTP keep-alive reduces the number of connections required to be opened. If the HTTP keep-alive is disabled, a new connection is made for every requested object on a web page.

5.1.1 Required Skills

- A working IIS Manager.

5.1.2 Symptoms

- Constant high load times for requested pages.
- Poor performance.

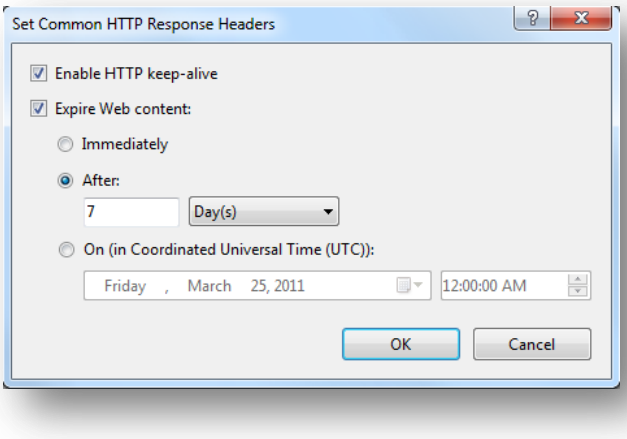
5.1.3 Procedure

To check if the HTTP keep-alive is enabled:

1. Launch **IIS Manager**.
2. Navigate to the site that you wish to check to see if the **HTTP keep-alive** is enabled.
3. Double click **HTTP Response Headers** — located in the IIS grouping.
4. In the **Actions** panel, click **Set common headers**...

5.1.4 Understanding the Results

The **Set Common HTTP Response Headers** dialog box appear. The following example shows that the HTTP keep-alive setting is enabled.



5.1.5 Sitecore Recommendation

Sitecore recommends that the IIS HTTP keep-alive is enabled.

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 © 2001-2013 Sitecore. All rights reserved.

5.1.6 How to Solve

To enable the HTTP keep-alive:

1. Launch **IIS Manager**.
2. Navigate to the site that you wish to enable the **HTTP keep-alive**.
3. Double click the **HTTP Response Headers** (located in the IIS grouping).
4. In the **Actions** panel, click **Set common headers**...
5. Select **Enable HTTP keep-alive**.
6. Click **OK**.

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 © 2001-2013 Sitecore. All rights reserved.

5.2 IIS Expire Web Content Header

The Expire Web content header (located in common HTTP Response headers) is how IIS determines whether or not to return a new version of the requested web page if the request is made after the web-page content has expired. IIS marks each web page before it's sent using the settings you provide for content expiration. The end-user's browser translates the expiration mark.

By setting Expire Web content to something other than immediately, you can reduce second-access load times by 50 to 70 percent. This setting does not affect dynamically generated content.

Please note that the following procedures are for IIS7.x. For information about how to enable the Expire Web content header in previous versions, refer to Microsoft's documentation.

5.2.1 Required Skills

- A working IIS Manager.

5.2.2 Symptoms

- Constant high load times for requested pages.
- Poor performance.

5.2.3 Procedure

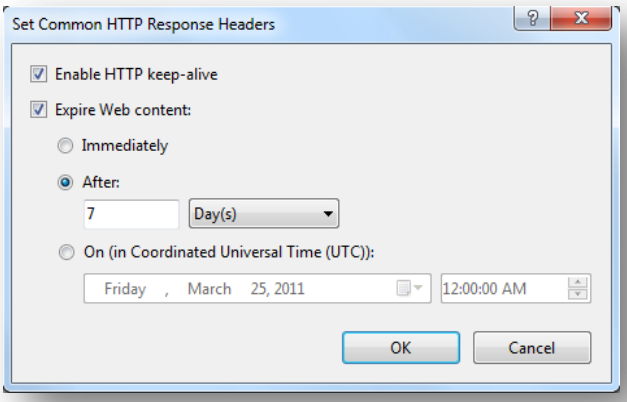
To check if the Expire Web content Header is enabled:

1. Launch **IIS Manager**.
2. Navigate to the site that you wish to check and see if the **Expire Web content header** is enabled.
3. Double click the **HTTP Response Headers** (located in the IIS grouping).
4. In the **Actions** panel, click **Set common headers**...

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 © 2001-2013 Sitecore. All rights reserved.

5.2.4 Understanding the Results

The **Set Common HTTP Response Headers** dialog box appears. The following example shows that the Expire Web content is enabled and set to **After** 7 days.



5.2.5 Sitecore Recommendation

Sitecore recommends that the Expire Web content be enabled, Set to "After" 30 days.

5.2.6 How to Solve

To enable the Expire Web content Header:

1. Launch **IIS Manager**.
2. Navigate to the site that you wish to enable the **Expire Web content heade**r.
3. Double click the **HTTP Response Headers** (located in the IIS grouping).
4. In the Actions panel, click **Set common headers**...
5. Select **Expire Web content**.
6. Select **After**.
7. Set the number of days to 30.
8. Click **OK**.

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 © 2001-2013 Sitecore. All rights reserved.

5.3 Enable IIS Static Content Compression

Enabling IIS static content compression static responses can be compressed and cached on disk across multiple requests, without degrading CPU resources.

By default, static content compression is enabled in IIS. This task is designed as a check to see if it is in fact enabled.

5.3.1 Required Skills

- A working IIS Manager.

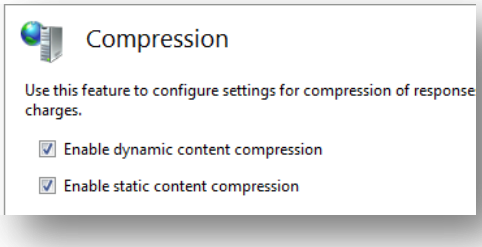
5.3.2 Symptoms

- Constant high load times for requested pages.
- Poor performance.

5.3.3 Procedure

To check if Static content compression is enabled:

1. Launch **IIS Manager**.
2. Navigate to the site that you wish to check and see if **Static content compression** enabled.
3. Double click the **Compression** icon (located in the IIS grouping).
4. The **Compression** dialog box appears. The following example shows that static content compression is enabled.



5.3.4 Sitecore Recommendation

Sitecore recommends that you enable the **IIS Static content compression** setting.

5.3.5 How to Solve

To enable IIS Static content compression:

1. Launch **IIS Manager**.
2. Navigate to the site that you wish to check and see if **Static content compression** enabled.
3. Double click the **Compression** icon (located in the IIS grouping).

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 © 2001-2013 Sitecore. All rights reserved.

4. Select the **Enable static content compression** check box.
5. Click **Apply**.

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 © 2001-2013 Sitecore. All rights reserved.

5.4 Enable IIS Dynamic Content Compression (Optional)

Enabling dynamic content compression is labeled as optional, because its benefits depend on the availability of CPU resources.

Dynamic content compression compresses responses for dynamic content, reducing bandwidth requirements. Because dynamic content is compressed with every request / response, there is a trade-off of increased CPU utilization, so for systems that are already running at a high CPU utilization level dynamic content compression should not be enabled.

A couple of interesting articles about the benefits of using dynamic compression, as well as how to set it up, can be found at:

- http://weblogs.asp.net/owscott/archive/2009/02/22/iis-7-compression-good-bad-how-much.aspx
- http://www.west-wind.com/weblog/posts/2011/May/05/Builtin-GZipDeflate-Compression-on-IIS-7x

5.4.1 Required Skills

- A working IIS Manager.

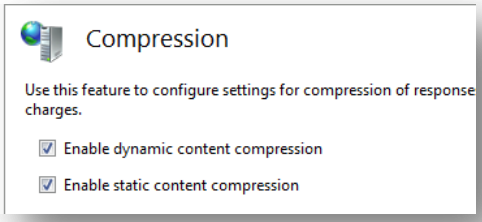
5.4.2 Symptoms

- Constant high load times for requested pages.
- Poor performance.

5.4.3 Procedure

To check if Dynamic content compression is enabled, do the following:

1. Launch **IIS Manager**.
2. Navigate to the site that you wish to check and see if **Dynamic content compression** enabled.
3. Double click the **Compression** icon (located in the IIS grouping).
4. The **Compression** dialog box appears. The following example shows that the dynamic content compression is enabled.



5.4.4 Sitecore Recommendation

Sitecore recommends that **IIS Dynamic content compression** be enabled on systems where CPU utilization is not at a constant high level.

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 © 2001-2013 Sitecore. All rights reserved.

5.4.5 How to Solve

To enable IIS Dynamic content compression:

1. Launch **IIS Manager**.
2. Navigate to the site that you wish to check and see if **Dynamic content compression** enabled.
3. Double click the **Compression** icon (located in the IIS grouping).
4. Select the **Enable dynamic content compression** check box.
5. If the **Enable dynamic content compression** check box is grayed out, you need to add the dynamic compression module to IIS by going to **Control Panel**, **Programs and Features**, **Turn Windows features on or off**
6. Click **Apply**.

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 © 2001-2013 Sitecore. All rights reserved.

Chapter 6

Tuning Procedures — Sitecore Client Optimizations

Tuning Procedures – Sitecore Client Optimizations contains a series of tasks designed to check the configuration, and recommended practices for the Sitecore client tools.

Proper configuration, and the following of recommended practices, help insure that the Sitecore users experience optimal performance while using the Sitecore client tools.

This chapter contains the following sections:

- Check Long Running Validators
- Check Excessive Item Versions
- Check Excessive Items per Node
- Miscellaneous Client Specific Optimizations

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 © 2001-2013 Sitecore. All rights reserved.

6.1 Check Long Running Validators

Validators that take a long time to run can have a negative performance impact while using the Sitecore Content Editor. This task is designed to look for those validation rules in question.

6.1.1 Required Skills

- A working knowledge of reading the Sitecore logs.

6.1.2 Symptoms

- Slow performance while using the Sitecore Content Editor.

6.1.3 Procedure to Check Long Running Validation Rules

Long running validation rules appear in the Sitecore logs with the following message: *Long running operation: Running Validation Rules*. To search for these messages you can either search for the above string using your favorite text editor, or by parsing the log file(s) with Sitecore Log Analyzer tool.

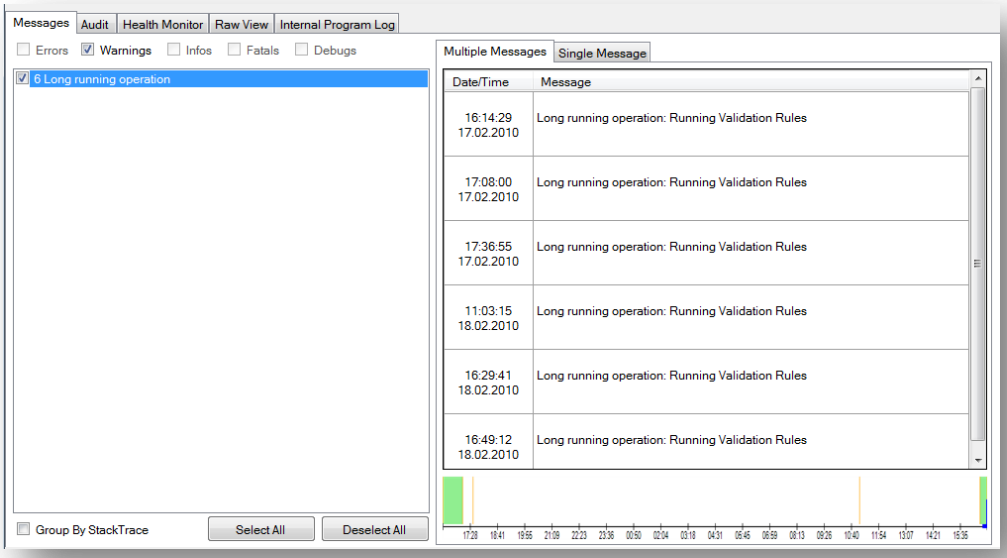
To parse the log file(s) using Sitecore Log Analyzer:

1. Launch **Sitecore Log Analyzer**.
2. Use **Select Location** button to select log file(s) for parsing.
3. Set **String filter** field to *Long running operation: Running Validation Rules* value.
4. Press **Analyze / Refresh*** button.
5. Go to **Messages** tab and check **Warnings** checkbox.
6. Check the checkbox of the "* *Long running operation*" entry in the list of messages.

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 © 2001-2013 Sitecore. All rights reserved.

6.1.4 Understanding the Results

The output looks like this:



The above table shows that 6 warnings were produced due to long running validation rules. Unfortunately the log files do not give an indication of which validation rules are producing the warnings.

6.1.5 Sitecore Recommendation

Sitecore recommends long running validation rules be investigated and disabled to help increase performance while using the Sitecore Content Editor.

6.1.6 How to Solve

The process of finding out which validation rules are producing warnings is a bit trial and error. Here are some steps that can be taken to help identify those validation rules that could be the culprits:

1. Look at any custom validation rules to see they are taking a long time to process. If so, disable them.
2. Disable any standard validators, by removing them from the *validation* rules section of an item, that are not required.
3. Some validators are declared in the `web.config` file (for example, the Media library validator).

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 © 2001-2013 Sitecore. All rights reserved.

6.2 Check Excessive Item Versions

Keeping around excessive numbers of versions for an item or items can have a negative impact on performance. Especially performance related to viewing the content tree in the content editor, as well as affecting the amount of memory that is utilized by caching.

As the numbers of versions grow, so does the amount of time required to process all of the versions. For example, when opening up an item to be viewed or edited within the content editor not only is the version that you are looking at being processed, but so are all of the other versions related to that item.

This task is designed to look for excessive versioning, as well as a pointer to a shared source module to manage versions.

6.2.1 Required Skills

- A working knowledge the Sitecore content editor.
- Experience installing and configuring shared source modules.

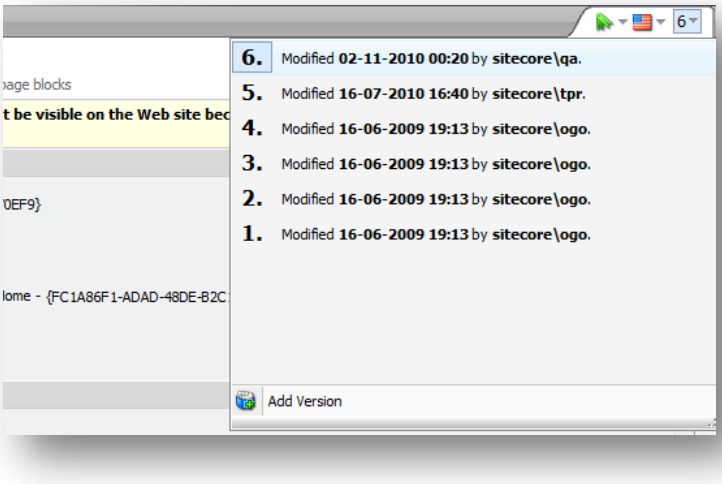
6.2.2 Symptoms

- Slow performance while using the Sitecore Content Editor.

6.2.3 Procedure to check excessive Item versions

Excessive numbers of versions is usually found on items that are the most heavily modified (for example, the home page of a site). Talk with the content authors to find out which pages are seeing the most activity — modification.

1. Launch the **Sitecore Content Editor**.
2. In the content tree, navigate to the items that are in question.
3. In the upper right hand corner of the item, there is a drop down showing the number of versions for the item being viewed:



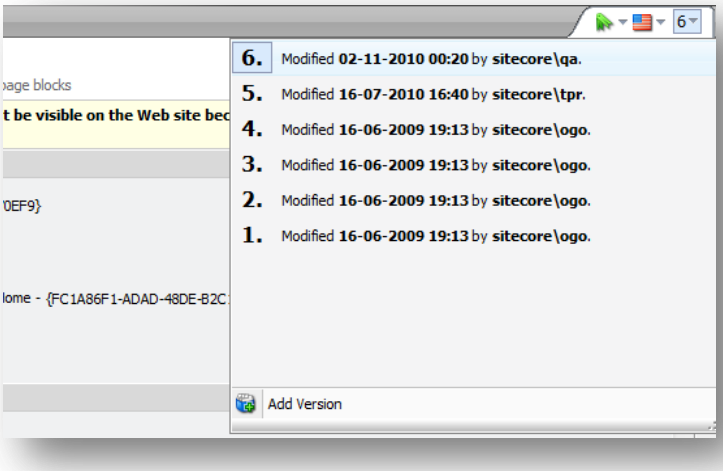
4. Note the number of versions for the item (anything over 10 should be flagged as excessive).

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 © 2001-2013 Sitecore. All rights reserved.

5. Repeat the process for other items in question.

6.2.4 Understanding the Results

The output looks like this:



This screen shows that there are currently 6 versions for this *Home* page being stored.

6.2.5 Sitecore Recommendation

Sitecore recommends that you store 10 or fewer versions of a particular item. However, this number may be higher based on company policy that dictates that number of required versions to be stored. Also, Sitecore recommends running the Version Manager shared source module to maintain the number of stored versions.

6.2.6 How to Solve

To manage the number of versions stored, there is a Version Manager shared source available at: http://trac.sitecore.net/VersionManager

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 © 2001-2013 Sitecore. All rights reserved.

6.3 Check Excessive Items per Node

The structure of the content tree should be well balanced without having an excess number of items under any parent node. This number should not exceed 100, so careful planning is required to provide structure through the use of folders for organization.

Excessive numbers of items, beyond 100, under any parent node decreases the performance perspective for a Sitecore user navigating the content tree in the Content Editor.

This task is designed to look for excessive items in the content tree.

6.3.1 Required Skills

- A working knowledge the Sitecore content editor.

6.3.2 Symptoms

- Slow performance while using the Sitecore Content Editor.

6.3.3 Procedure to Check for Excessive Items per Node

Searching for nodes that have in excess of 100 items can be either treated as a manual task, or a script can be created to automate the process. To manually search for nodes with an excessive number of items:

1. Launch the **Sitecore Content Editor**.
2. In the content tree, expand all nodes.
3. Look for any node that has in excess of 100 items.

   To search for nodes with an excessive number of children with a script, create a script that recursively traverses the content tree from a given home node. For example:

```
        using System;
        using System.Collections.Generic;
        using System.Linq;
        using System.Web;
        using System.Web.UI;
        using System.Web.UI.WebControls;
        using Sitecore.Data.Items;
        using Sitecore.Collections;

        namespace Sitecore.Utilities
        {
          public partial class NodeCount : System.Web.UI.Page
          {
            protected void Page_Load(object sender, EventArgs e)
            {
              Item homeItem =
Sitecore.Configuration.Factory.GetDatabase("master").Items["/sitecore/content/home"];
              getNumberOfChildren(homeItem);
            }

            private void getNumberOfChildren(Item node)
            {
              ChildList children = new ChildList(node);
              foreach (Item child in children)
              {
                if (child.HasChildren && child.Children.Count >= 100)
```

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 © 2001-2013 Sitecore. All rights reserved.

```
                {
                  Response.Write("Node: " + child.Name + " - " + child.Children.Count + "<br
/>");
                  getNumberOfChildren(child);
                }
                else
                {
                  Response.Write("<br />");
                }
              }
            }
          }
        }
```

6.3.4 Understanding the Results

The results depend on the number of nodes found with greater than 100 items. Any node that has greater than 100 items should be flagged as needing attention.

6.3.5 Sitecore Recommendation

Sitecore recommends that 100 or fewer items under any parent node.

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 © 2001-2013 Sitecore. All rights reserved.

6.4 Miscellaneous Client Specific Optimizations

Presented here are a series of optimizations that improve performance for the Sitecore clients.

This task explains how to add this setting to the `web.config` file.

6.4.1 Required Skills

- A working knowledge of the Sitecore web.config file.

6.4.2 Symptoms

- Slow performance while using the Sitecore Content Editor.

6.4.3 ContentEditor.RenderCollapsedSections setting

The `ContentEditor.RenderCollapsedSections` setting is a hidden setting in the `/web.config file`, which by default is true. Adding `<setting name="ContentEditor.RenderCollapsedSections" value="false" />` to the `/web.config` file improves client performance, especially with large content trees.

Also, setting the value to false prevents the prefetch caching from adding specified children into memory twice.

Do Not Show Standard Fields in the Content Editor.

By disabling the viewing of Standard Fields in the Content Editor, performance improves because Sitecore does not have to render as much information per item.

6.4.4 Make Sure that Content Authors Never Use Full Publish

Full Publish, or publishing all items, is very resource intensive. During at full publish, performance degradation is possible for any Sitecore user currently working on the system during the operation. It is highly recommended that Smart or Incremental publishing be used instead.

6.4.5 ContentEditor.CheckHasChildrenOnTreeNodes setting

This setting determines if the editor uses the `HasChildren` method when rendering a tree node. Set this value to false to increase performance.

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 © 2001-2013 Sitecore. All rights reserved.

Chapter 7

Tuning Procedures — Sitecore Miscellaneous

Content Delivery Server Optimizations

This chapter contains a series of tasks designed to check the configuration, and recommended practices for the content delivery server(s).

The optimizations shown in this chapter are designed to disable functionality that is typically used for diagnostics, and/or content management functionality. By disabling these features, CPU process time is recovered.

Proper configuration, and the following of recommended practices, helps insure that the Sitecore content delivery server(s) run at their optimal performance levels.

This chapter contains the following sections:

- Disable WebDAV
- Disable Performance Counters
- Disable Memory Monitor

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 © 2001-2013 Sitecore. All rights reserved.

7.1 Disable WebDAV

WebDAV allows users of Web clients such as browsers to manage files on Web servers using Web protocols such as HTTP or HTTPS. With WebDAV, users can drag-and-drop files between the Windows desktop and the browser.

Disabling WebDAV on the content delivery server(s) makes sense as it is not expected to be used in these environments.

7.1.1 Required Skills

- A working knowledge of Sitecore configuration files.

7.1.2 Symptoms

- Unnecessary log files being created on the content delivery server.

7.1.3 Procedure to Disable WebDAV

To disable WebDAV:

1. Open the `web.config` file in an editor — located in the web root directory.
2. Navigate to `<log4net/>` section and remove the WebDAV references.

```
  <log4net>
  <appender name="LogFileAppender" type="log4net.Appender.SitecoreLogFileAppender,
Sitecore.Logging">
  <file value="$(dataFolder)/logs/log.{date}.txt" />
  <appendToFile value="true" />
  <layout type="log4net.Layout.PatternLayout">
  <conversionPattern value="%4t %d{ABSOLUTE} %-5p %m%n" />
  </layout>
  </appender>
  <!--<appender name="WebDAVLogFileAppender" type="log4net.Appender.SitecoreLogFileAppender,
Sitecore.Logging">
  <file value="$(dataFolder)/logs/WebDAV.log.{date}.txt" />
  <appendToFile value="true" />
  <layout type="log4net.Layout.PatternLayout">
  <conversionPattern value="%4t %d{ABSOLUTE} %-5p %m%n" />
  </layout>
  </appender>-->
  <root>
  <priority value="INFO" />
  <appender-ref ref="LogFileAppender" />
  </root>
  <!--<logger name="Sitecore.Diagnostics.WebDAV" additivity="false">
  level value="INFO" />
  <appender-ref ref="WebDAVLogFileAppender" />
  </logger>-->
  </log4net>
```

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 © 2001-2013 Sitecore. All rights reserved.

3. Navigate to `<system.webServer/>` section and remove the WebDAV references.

```
   <!--<remove name="WebDAVModule" />-->
   <!–
<add name="WebDAVRoot" path="*" verb="OPTIONS,PROPFIND" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll"
resourceType="Unspecified" preCondition="classicMode,runtimeVersionv2.0,bitness32" />
   <add name="WebDAVRoot64" path="*" verb="OPTIONS,PROPFIND" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework64\v2.0.50727\aspnet_isapi.dll"
resourceType="Unspecified" preCondition="classicMode,runtimeVersionv2.0,bitness64" />
   <add verb="*" path="sitecore_webDAV.ashx"
type="Sitecore.Resources.Media.WebDAVMediaRequestHandler, Sitecore.Kernel"
name="Sitecore.WebDAVMediaRequestHandler" />
-->
```

4. Navigate to `<httpHandlers/>` section and remove the WebDAV references.

```
   <!--
   <add verb="*" path="sitecore_webDAV.ashx"
type="Sitecore.Resources.Media.WebDAVMediaRequestHandler, Sitecore.Kernel" />
   -->
```

5. Save your changes.
6. Finally, you can delete the `Sitecore.WebDAV.config` file from the `webroot/App_Config/Include` directory.

7.1.4 Sitecore Recommendation

Sitecore recommends disabling WebDAV on the production content delivery servers to reduce the number of log files being created. Also, Sitecore recommends disabling WebDAV on the content management servers if the WebDAV functionality is not being used.

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 © 2001-2013 Sitecore. All rights reserved.

7.2 Disable Performance Counters

Performance counters create a minor overhead and is recommended to be enabled only when running in troubleshooting mode.

7.2.1 Required Skills

- A working knowledge of Sitecore configuration files.

7.2.2 Symptoms

- Slight overhead required to maintain the performance counters.

7.2.3 Procedure to Disable Performance Counters

To disable Performance Counters:

1. Open the `web.config` file in an editor — located in the web root directory.
2. Navigate to `<setting name="Counters.Enabled" value="true" />` section and set the value to false.

```
<setting name="Counters.Enabled" value="false" />
```

3. Save your changes.

7.2.4 Sitecore Recommendation

Sitecore recommends disabling Performance Counters on the production content delivery servers to reduce the overhead required to maintain the counters.

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 © 2001-2013 Sitecore. All rights reserved.

7.3 Disable Memory Monitor

Memory Monitor creates frequent CPU spikes and should only be enabled when troubleshooting memory related issues.

7.3.1 Required Skills

- A working knowledge of Sitecore configuration files.

7.3.2 Symptoms

- Frequent CPU spikes.

7.3.3 Procedure to Disable Memory Monitor

To disable WebDAV:

1. Open the `web.config` file in an editor (located in the web root directory).
2. Navigate to `<hooks/>` section and comment out the `<hook type="Sitecore.Diagnostics.MemoryMonitorHook, Sitecore.Kernel" />`.

```
      <hooks>
     <hook type="Sitecore.Diagnostics.HealthMonitorHook, Sitecore.Kernel" />
     <!--<hook type="Sitecore.Diagnostics.MemoryMonitorHook, Sitecore.Kernel">
       <param desc="Threshold">800MB</param>
       <param desc="Check interval">00:00:05</param>
       <param desc="Minimum time between log entries">00:01:00</param>
       <ClearCaches>false</ClearCaches>
       <GarbageCollect>false</GarbageCollect>
       <AdjustLoadFactor>false</AdjustLoadFactor>
     </hook>-->
   </hooks>
```

3. Save your changes.

7.3.4 Sitecore Recommendation

Sitecore recommends disabling the Memory Monitor in production environments, and only enabling it for troubleshooting memory related issues.

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 © 2001-2013 Sitecore. All rights reserved.

Chapter 8

Tuning Procedures — Sitecore Search

This chapter contains a series of tasks designed to check the configuration and recommended practices for Sitecore Search on content delivery servers.

The optimizations shown in this chapter are designed to disable functionality that is typically used for diagnostics. By disabling these features, CPU process time is recovered and overhead is reduced.

Proper configuration, and the following of recommended practices, helps ensure that Sitecore content delivery servers and Sitecore-based Search run at their optimal performance levels.

This chapter contains the following sections:

- Disable Bucket Debug
- More Information

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 © 2001-2013 Sitecore. All rights reserved.

8.1 Disable Bucket Debug

If bucket debugging is enabled, every query executed against an item bucket is written to the search log file.

8.1.1 Required Skills

- A working knowledge of Sitecore configuration files.

8.1.2 Symptoms

- Large search log files filled with search queries

8.1.3 Procedure to Disable Bucket Debug

- Open the `App_Config/Include/Sitecore.Buckets.config` file in an editor.
- Set the `BucketConfiguration.EnableBucketDebug` setting to false:

```
<setting name="BucketConfiguration.EnableBucketDebug" value="false" />
```

- Save your changes.

8.1.4 Sitecore Recommendation

Sitecore recommends disabling bucket debug logging on content delivery servers to reduce log file size, mitigate pollution of the log files, and help minimize any overhead required to write and maintain log files.

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 © 2001-2013 Sitecore. All rights reserved.

8.2 More Information

For more information about Sitecore Search and how to properly configure search operations for your environment, see the following documents on SDN:

- Developer's Guide to Item Buckets and Search
- Sitecore Search Scaling Guide
- Sitecore Search Admin Guide

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 © 2001-2013 Sitecore. All rights reserved.