Sitecore CMS 6.0 - 6.4

CMS Performance Tuning Guide

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

Table of Contents

Chapter 1 Introduction 3
Chapter 2 Diagnostic Procedures 4
2.1 Browser Page Test 5
2.2 Investigating Page Performance (using the IIS Logs) 14
2.3 Rendering Performance 18
2.4 Investigating Memory Leaks with the Sitecore Logs 21
Chapter 3 Tuning Procedures - General 33
3.1 Check SQL Server Index Fragmentation Level 34
3.2 Check for SQL Server Maintenance Plan 37
3.3 Cleanup Database Tables 41
3.4 Check Database Cleanup Agents 44
3.5 Disable Search Indexes if Not Used 48
3.6 Software and Server Configuration 50
Chapter 4 Tuning Procedures - Database Properties 61
4.1 Compatibility Level Set To SQL Server 2008 (100) 62
4.2 Auto Close Property Set To False 65
4.3 Auto Shrink Property Set To False 68
4.4 Recovery Model Set to Simple 71
Chapter 5 Tuning Procedures - Sitecore Caches 74
5.1 Setting Initial Cache Values 75
5.2 Tuning Sitecore Caches 79
5.3 Configuring Prefetch Cache Guidelines 82
5.4 Configuring Output (Rendering) Cache Guidelines 84
Chapter 6 Tuning Procedures - IIS Settings 88
6.1 Enable IIS HTTP keep-alive 89
6.2 IIS Expire Web Content Header 92
6.3 Enable IIS Static Content Compression 95
6.4 Enable IIS Dynamic Content Compression (Optional) 97
Chapter 7 Tuning Procedures - Sitecore Client Optimizations 100
7.1 Check Long Running Validators 101
7.2 Check Excessive Item Versions 104
7.3 Check Excessive Items per Node 107
7.4 Miscellaneous Client Specific Optimizations 110

Chapter 1

Introduction

This guide is designed in two parts. The first being a series of diagnostic procedures used to aid in the discovery of performance issues, as well as a means to measure performance gains through tuning. And the second being a series of tuning tasks, broken out by relative sections, that helps insure that the Sitecore implementation is healthy and running at its optimal performance.

This manual contains the following chapters:

- **Chapter 1 —Introduction**
- **Chapter 2 — Diagnostic Procedures**
- **Chapter 3 — Tuning Procedures - General**
- **Chapter 4 — Tuning Procedures - Database Properties**
- **Chapter 5 — Tuning Procedures - Sitecore Caches**
- **Chapter 6 — Tuning Procedures - IIS Settings**
- **Chapter 7 — Tuning Procedures - Sitecore Client Optimizations**

Chapter 2

Diagnostic Procedures

Diagnostic procedures are a series of tests that help identify performance issues in a Sitecore implementation.

The procedures defined are complimentary to the CMS tuning. By running the diagnostic procedures before and after tuning the Sitecore CMS, performance improvements can be recorded.

This chapter contains the following sections:

- Browser Page Test
- Investigating Page Performance (using the IIS Logs)
- Rendering Performance
- Investigating Memory Leaks with the Sitecore Logs

2.1 Browser Page Test

"The size of the average web page of the top 500 websites has more than quintupled since 2003. From 2003 to 2009 the average web page grew from 93.7K to over 507K (see Figure 1), over 5.4 times larger (Domenech et al. 2007, Flinn & Betcher 2008, Charzinsk 2010). During the same six-year period, the number of objects in the average web page more than doubled from 25.7 to 64.7 objects per page. Longer term statistics show that since 1995 the size of the average web page has increased by 35 times, and the number of objects per page has grown by 28 times." - http://www.websiteoptimization.com/speed/tweak/average-web-page/

What the previous statement means, is that in today's webpages it is crucial to understand the number and size of objects that are loaded into a webpage. Also, how a web server is setup to cache objects and minimize the number of requests that are made is important.

There is a lot of information that can be gathered by analyzing what takes place when a webpage is loaded into a browser. Information about what is being requested, time to response, size of objects, and so on is available. Also how the server is setup in terms of caching, compression, CDN usage, and keep-alive information can be viewed.

This task looks at using an open source plug-in, AOL Pagetest, along with how to interpret the results.

2.1.1 Setup

The AOL Pagetest browser plug-in can be downloaded from:
http://sourceforge.net/projects/pagetest/files/

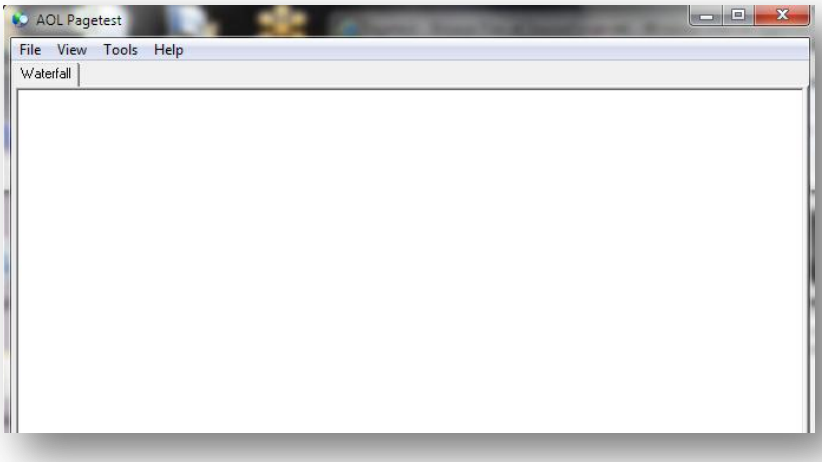
Once it has been downloaded, install in its default location. The AOL Pagetest plug-in works with IE 7+ browsers, and appears in the Tools menu.

Note
If IE was opened up during the install, you must restart IE for the AOL Pagetest to appear in the Tools menu.

2.1.2 How to Use the AOL Pagetest Browser Plug-in

To launch AOL Pagetest:

1. Launch IE.
2. In the **Tools** menu, *click* **AOL Pagetest** .



2.1.3 Capturing Information

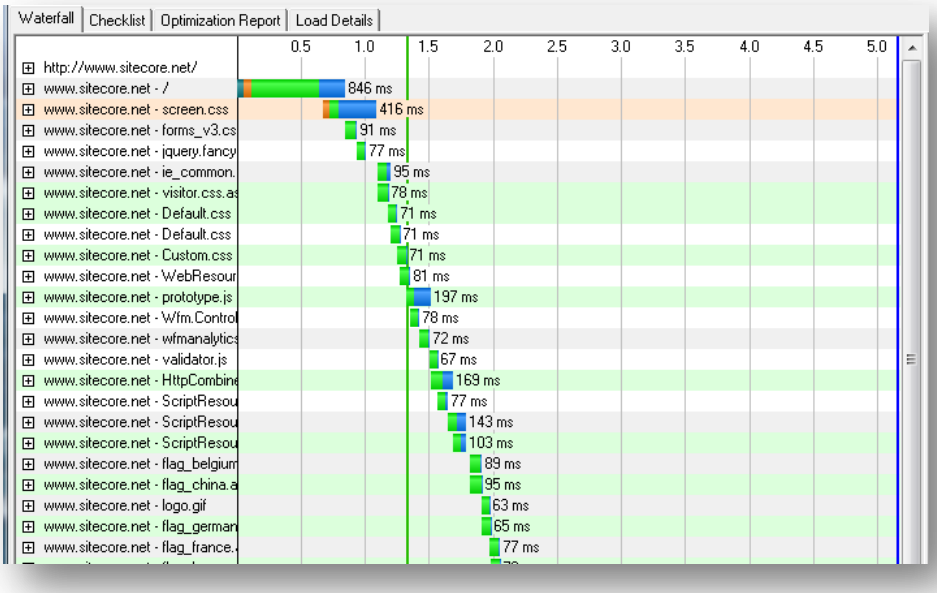
There are 3 useful page loading scenarios that are gathered during this exercise to see how the webpage reacts to a cleared browser cache, a webpage reload, and a webpage request with a loaded cache.

- Cleared browser cache - This simulates the first time a page has been visited from the browser being used.
  - To clear the browser cache for **IE 8** and above:
  - In the **Safety** menu, click Delete **Browsing History...**
  - Clear **Preserve Favorites website data**, and select T**emporary Internet Files**, **Cookies**, and **History**
- To clear the browser cache for **IE 7**:
  - In the **Tools** menu, click **Internet Options**
  - Under **Browsing History**, click **Delete**...
  - To delete your cache, click **Delete files**...
  - Click **Close**, **OK**.
- Webpage reload or refresh — This forces requests to be made to objects in cache, resulting in 304 status codes. When a 304 status appears, a request is still made for an object, even though no download occurs.
- Accessing a recently visited webpage (typing in the URL) — This results in a page with a much reduced request chain, since items that were previously loaded into the browser cache are accessed without making any requests.

First Pass — Cleared Browser *C*ache

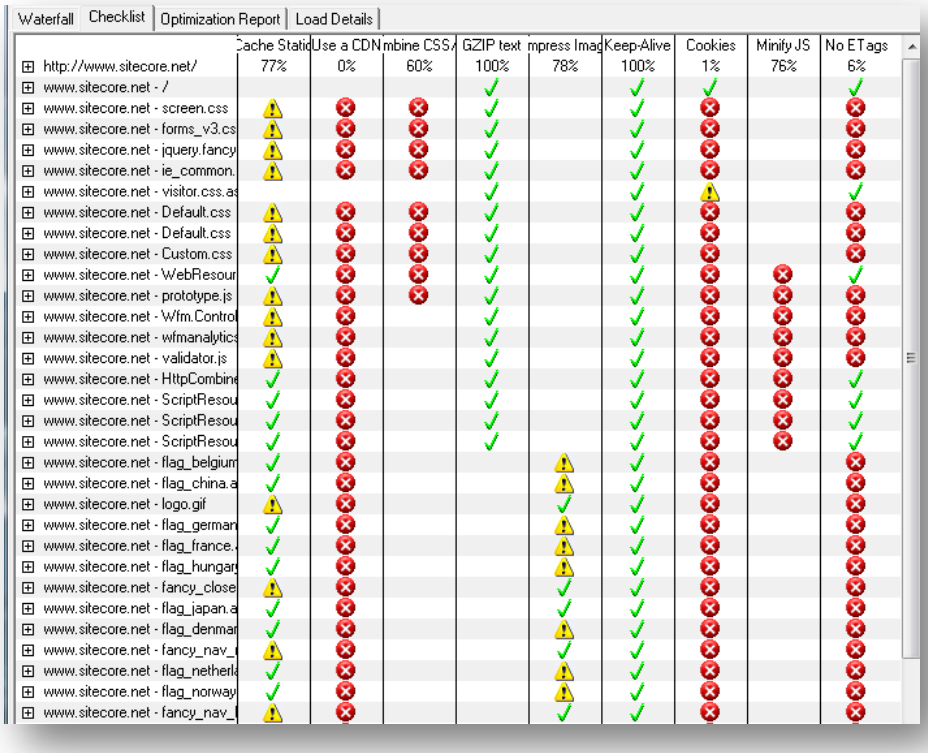
1. Launch ***IE*.**
2. Clear the browser cache (see above).
3. Launch AOL Pagetest.
4. Type in the URL you wish to test (for example: http://www.sitecore.net).
5. The AOL Pagetest window now contains 4 tabs (**Waterfall**, **Checklist**, **Optimization Report**, and **Load Details**). The following section explains how to interpret the information presented in each tab.

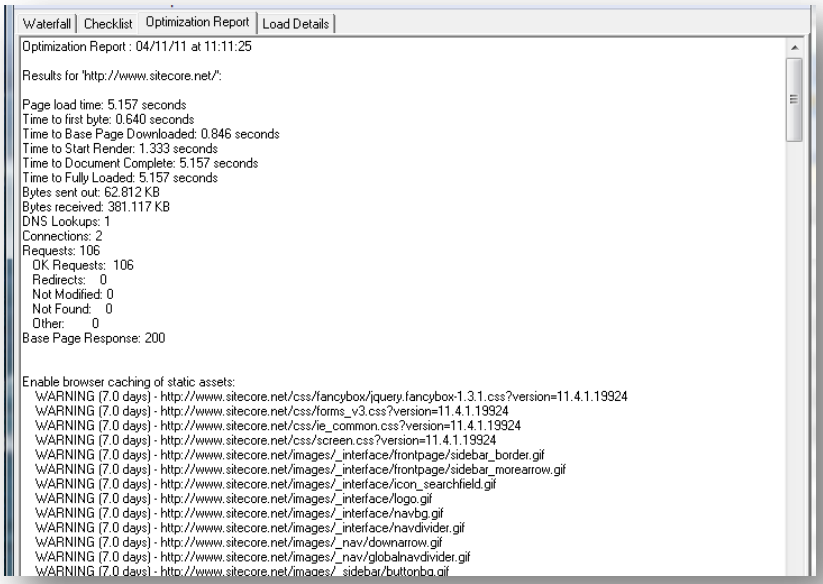
A view of the 4 tabs:



Observations:

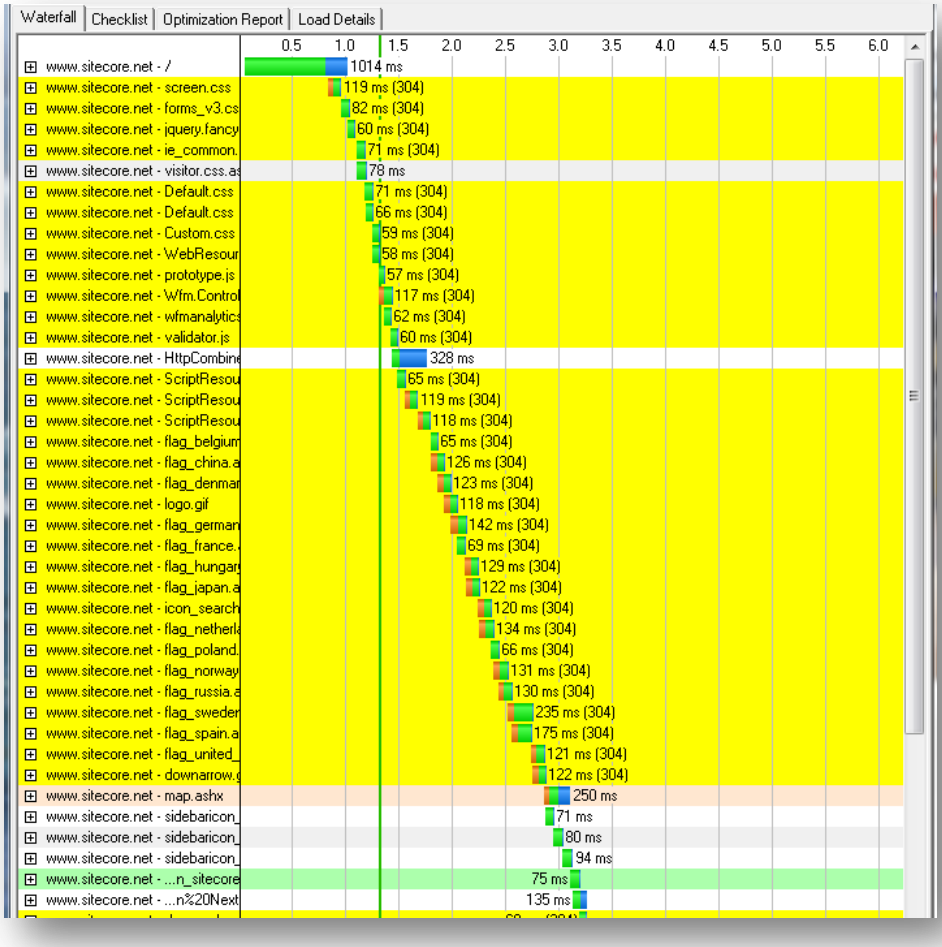
- Notice that the **Waterfall** tab shows a full request made to each object in the webpage.
- That assets such as CSS and JS files have not been combined, so multiple requests are being made to download them. By combining all of your CSS and JS files, the time line is pushed to the left, due to a single request being made, shortening the time to render the page.





Second Pass - Refresh Browser Page

1. In the **AOL Pagetest File** menu, click **New**
2. Refresh the browser page — click refresh page, or press F5.
3. View the results in the **Waterfall** tab, and compare to the results from the first pass.



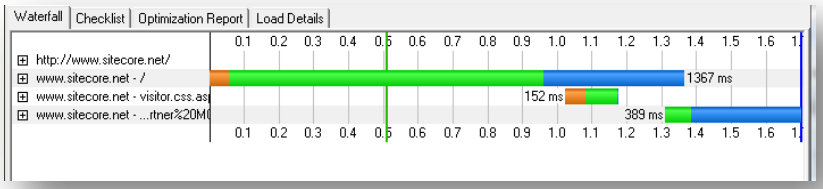
Observations:

- The objects highlighted in yellow are 30x status codes.
- The results show that cached items are requested, but not downloaded. This is expected behavior for a page that has been refreshed.
- What can be taken from this view is that those items with a 304 status code should not be requested during the 3rd pass, "A recently visited page".

Third Pass — Recently Visited Webpage

1. In the **AOL Pagetest File** menu, click **New**.
2. In the browser select the URL and hit enter (do not click the browser's refresh button).

3. View the results in the **Waterfall** tab, and compare to the results from the first and second passes.



Observations:

- The request tree has been reduced significantly.
- It is known that the first two requests, / and visitor.css, are dynamic and not cached.
- The third item requires investigation as to why it is not coming from cache. Upon further investigation it was found that this request is a rotating image, which changes across visits. (OK not to be in cache).

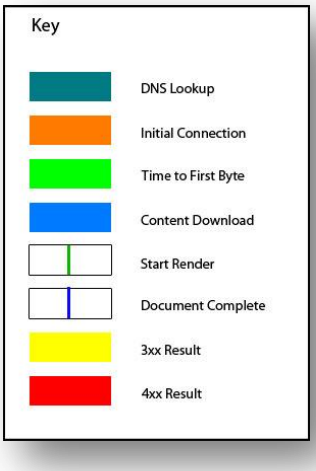
2.1.4 Interpreting the Results

Here we dive into the **Waterfall**, **Checklist**, **Optimization Report**, and **Load Details** tabs, explaining the information that is presented.

Waterfall Tab

The **Waterfall** tab presents a view of the request chain that makes up the requested objects for a web page. The requested objects are listed down the left side, and the time is listed along the top.

The colors that make up a request / response are:



- **DNS Lookup** — This is the time it takes to look up the IP Address for the requested URL.
- **Initial Connection** — This is the time that it takes to go from the client to the server to open up a socket. Note, if an initial connection is made on every request, this could be an indication that the HTTP keep-alive has not been enabled on the server.
- **Time to First Byte** — This is the time that it takes from request to receiving the first byte of the object. Extended Time to First Byte times could be an indicator of a problem at the server (i.e. performance issues), or network issues.

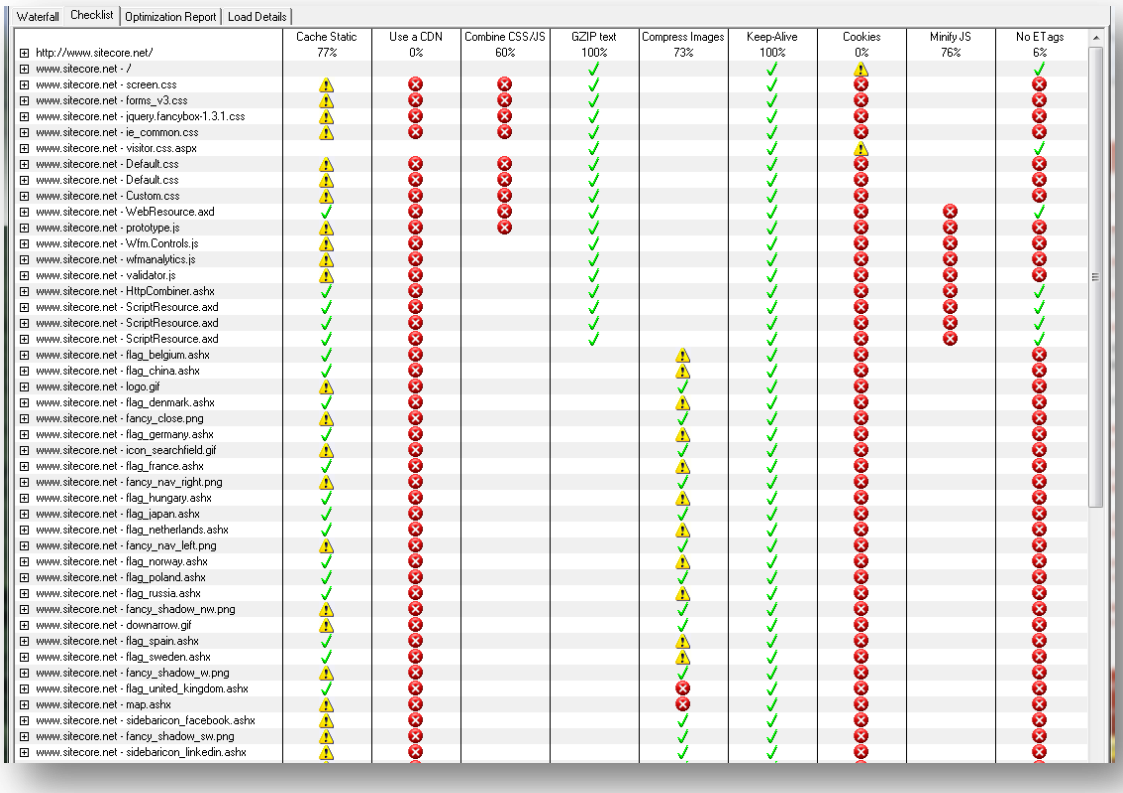
- **Content Download** — This is the time required to download an object. Extended Content Download times could be an indicator of large objects. An inspection of the size of the object may be required. For example: images that have not been optimized or compressed could have an effect on the performance of the web page.
- **Start Render** — This is the green vertical line shown on the Waterfall. This indicates when the user starts to see content appear in the browser. Technically this represents the time when the web page has a height and width > 0.
- **Document Complete** — This is the time when the document complete event had fired.
- **3xx Result** — Objects that are highlighted in yellow indicate that 3xx status code has occurred, such as a request or cache request.
- **4xx Result** — Objects that are highlighted in red indicate that a 4xx status, or error code has occurred. These should be investigated ASAP.

Checklist Tab

The **Checklist** tab is a report card of how well your site takes advantage of performance related settings and procedures.

Topics are listed from left to right in order of importance:

- A green check mark indicates a *pass.*
- A yellow triangle indicates a *warning.*
- A red X indicates a "failure"



The following table contains information about the meaning of each topic, what objects they apply to, as well as what is checked:

Cache Static	Applicable Objects	Any non-html object with a mime type of "text/*",

		"*javascript*" or "image/*" that does not explicitly have an Expires header of 0 or -1, a cache-control header of "private", "no-store" or "no-cache" or a pragma header of "no-cache"
	What is checked	An "Expires" header is present (and is not 0 or -1) or a "cache-control: max-age" directive is present and set for an hour or greater. If the expiration is set for less than 30 days you get a warning (only applies to max-age currently).
Use A CDN	Applicable Objects	All static non-html content (css, js and images)
	What is checked	Checked to see if it is hosted on a known CDN (CNAME mapped to a known CDN network). The known CDN's are Akamai, Amazon CloudFront, Coral Cache, Edgecast, Google, Highwinds, Internap, Limelight, Mirror Image, Panther and Yahoo
Combine CSS/JS	Applicable Objects	All css and javascript objects
	What is checked	If multiple files of the same type are served then each additional css file beyond 1 subtracts 5 percent and each Javascript file beyond the first subtracts 10 percent
GZIP Text	Applicable Objects	All objects with a mime type of "text/*" or "*javascript*"
	What is checked	Transfer-encoding is checked to see if it is gzip. If it is not then the file is compressed and the percentage of compression is the result (so a page that can save 30% of the size of it's text by compressing would yield a 70% test result)
Compress Images	Applicable Objects	Any object with a mime type of "image/*"
	What is checked	GIF - All pass PNG - Must be 8 bit or lower (no 24-bit PNGs pass) JPEG - Within 10% of a photoshop quality 50 pass, up to 50% larger warns and anything larger than that fails. The overall score is the percentage of image bytes that can be saved by re-compressing the images.
Keep-Alive	Applicable Objects	All objects that are from a domain that serves more than one object for the page — for example, if only a single object is served from a given domain it is not checked.
	What is checked	The response header contains a "keep-alive" directive or the same socket was used for more than one object from the given host
Cookies	Applicable Objects	All requests
	What is checked	Any request for a static object that sends up a cookie fails. All other requests that send up cookies warn.
Minify JS	Applicable Objects	All html, javascript and json responses
	What is checked	Javascript is run through jsmin. If the original content was gzip encoded, the minified version is also gzipped for comparison. If > 5KB or 10% is saved then it fails. If > 1KB is saved, it warns, otherwise it passes.
No Etags	Applicable Objects	All requests
	What is checked	If the response headers include an ETag header then the request fails.

Information to Solving Failures and Warnings:

- Failures in the Cache Statics column indicate that the HTTP Expires Web content header is not set, refer to: *IIS Expire Web Content Header*
- Warnings in the Cache Statics column indicate that the HTTP Expires Web content header has a value set to less than 30 days. Refer to: *IIS Expire Web Content Header*

- Failures in the Combine CSS /JS column indicate that CSS and or JS files are not combined into fewer files, indicating that multiple requests are being made to pull in these objects.
- Failures in the GZIP text column indicate that static compression is not enabled on the web server. Refer to*: Enable IIS Static Content Compression*
- Failures in the Keep-Alive column indicate that HTTP keep-alive is not enabled on the web server, causing a connection having to be initiated for every object requested. Refer to: *Enable IIS HTTP keep-alive*

Optimization Report Tab

The **Optimization Report** tab provides general information related to load times for the page, as well as where improvements can be made (based on the topics in the **Checklist** tab). In some cases, such as compression settings, an estimate of how much savings in terms of size is provided.

The **Optimization Report** tab is a textual view of what is presented in both the **Waterfall** and **Checklist** tabs.

Load Details Tab

The **Load Details** tab is a detailed account of what happens with every request. This information is useful for narrowing down issues with a particular request.

2.2 Investigating Page Performance (using the IIS Logs)

This Investigating Page Performance Task is designed to find pages that take longer than 4000 ms (4 sec) round trip, using the IIS logs. The results are recorded and used as a starting point for further investigation into which rendering component, or components, may be the culprit leading to poor performance.

The advantage of parsing information from the IIS logs vs. measuring page performance directly is that you also are able to look at geo-location information (possible network issues that occur due to location), and issues that may occur during peak usage (indicators of capacity problems).

2.2.1 Required Skills

- A working knowledge of Log Parser Lizard GUI
- A working knowledge of the IIS Logs.

2.2.2 Symptoms

- Slow page round trip times.

2.2.3 Talk to the Partner / Customer

The first step in determining where problem / slow loading pages exists is to ask the partner / customer / website owner if they are aware of any pages that appear to be slow. Record the URIs for further investigation into what is causing the performance issues.

Also, find out what are the peak run times for the web site. This information is used to determine if the web site has the capacity to handle load during peak usage.

2.2.4 Procedure to Query IIS Logs for Long Running Requests

This query looks at the requested URLs ordered by time required to process the requests in descending order. This allows you focus on a rendering, or set of renderings in which to investigate.

Note that we are removing any URLs that include /sitecore/ to avoid looking at Sitecore client tools traffic.

To query the IIS Logs, looking for long running requests:

1. Launch **Log Parser Lizard GUI**.
2. Create a **New Query**.
3. The log format by default is set to **IIS Logs** Input Format.

```
        Run the following script(change the  #logs location#):

SELECT TO_TIMESTAMP(date, time) as Timestamp, cs-uri-stem as [URI , c-ip AS [Client IP], time-
taken as [Time],sc-status as [Status]
FROM #logs location#
WHERE ((EXTRACT_EXTENSION(URI) = 'aspx' OR EXTRACT_EXTENSION(URI) = '') AND URI NOT LIKE
'%/sitecore%')
AND time-taken > 4000
AND Status = '200'
ORDER BY time-taken DESC
```

2.2.5 Understanding the Results

The results are shown in two parts. The first is a series of long running requests that are all coming from the same client IP address. This could be an indication of in-house testing. The second is a more typical set of results coming from external traffic to the site.

Results Part 1

Query

Timestamp	URI	Client IP	Time	Status
27.01.2011 16:24:40	/Sverige.aspx	88.131.15.18	567,640	200
30.01.2011 08:35:30	/en/Norge.aspx	195.184.101.130	468,644	200
30.01.2011 08:35:30	/Danmark.aspx	195.184.101.130	468,363	200
30.01.2011 08:35:31	/Sverige.aspx	195.184.101.130	467,082	200
30.01.2011 08:35:31	/Sverige.aspx	195.184.101.130	466,504	200
30.01.2011 08:35:32	/	195.184.101.130	451,616	200
30.01.2011 08:35:33	/en/Norge.aspx	195.184.101.130	445,305	200
30.01.2011 08:35:32	/Danmark.aspx	195.184.101.130	444,945	200
30.01.2011 08:35:34	/Sverige.aspx	195.184.101.130	443,945	200
30.01.2011 08:35:33	/Sverige.aspx	195.184.101.130	443,492	200
30.01.2011 08:35:36	/UnitedKingdom.aspx	195.184.101.130	437,650	200
30.01.2011 08:35:36	/	195.184.101.130	437,462	200
30.01.2011 08:35:35	/Danmark.aspx	195.184.101.130	436,869	200
30.01.2011 08:35:35	/UnitedKingdom.aspx	195.184.101.130	436,791	200
30.01.2011 08:35:34	/en/Norge.aspx	195.184.101.130	436,588	200
30.01.2011 08:35:36	/	195.184.101.130	422,122	200

To check to see where the Client IP address is geo-located, go to http://ip2location.com/1.2.3.4, where 1.2.3.4 is replaced with the client IP address. For example, http://ip2location.com/195.184.101.130 would yield (note: this is also useful for tracking down possible network issues based on location):

IP Address	Country	Region	City	Latitude/ Longitude	ZIP Code	Time Zone	
195.184.101.130	DENMARK	KOBENHAVN	COPENHAGEN	55.676294 12.568116	-	+01:00	Map It

Net Speed	ISP	Domain
COMP	RESULTMAKER A/S	RESULTMAKER.COM

IDD Code	Area Code	Weather Station
45	-	DAXX0009 - COPENHAGEN

MCC	MNC	Mobile Brand
-	-	-

Results Part 2

4 second threshold - IIS Logs

Timestamp	URI	Client IP	Time	Status
31.01.2011 10:13:48	/en/Company/Contact/Japan.aspx	173.203.158.156	5,170	200
31.01.2011 11:59:07	/News/RSS/Feeds/Denmark-News.aspx	193.3.234.5	5,155	200
30.01.2011 21:54:23	/Danmark.aspx	195.184.101.130	5,139	200
30.01.2011 11:54:36	/de/Hungary.aspx	78.46.71.246	5,121	200
27.01.2011 10:36:27	/en/Products/Resources/Tours.aspx	65.61.164.180	5,092	200
31.01.2011 10:17:45	/en/Customers/Selected-Customers.aspx	173.203.158.156	5,077	200
30.01.2011 11:57:48	/en/Solutions/Best-CMS-Solutions-Education.aspx	78.46.71.246	5,076	200
28.01.2011 01:09:32	/en/Japan.aspx	166.205.138.71	5,061	200
31.01.2011 10:14:02	/en/Products.aspx	173.203.158.156	5,046	200
29.01.2011 10:47:10	/en/Partners.aspx	65.61.143.45	5,030	200
27.01.2011 07:33:24	/products/resources/whitepapers/gartner-magic-quadrant.aspx	202.155.14.116	5,030	200
31.01.2011 10:14:10	/en/Partners/Hosting-Partners.aspx	173.203.158.156	4,999	200
31.01.2011 10:17:50	/en/Customers/Selected-Customers.aspx	173.203.158.156	4,983	200
28.01.2011 10:37:16	/en/Products/Industry-Commentary.aspx	64.39.4.224	4,983	200
29.01.2011 10:50:27	/en/Customers.aspx	65.61.143.45	4,983	200
31.01.2011 10:16:31	/Japan.aspx	173.203.158.156	4,983	200
30.01.2011 11:58:36	/en/News/NewsAndEvents.aspx	78.46.71.246	4,982	200
30.01.2011 11:56:21	/en/Company/Contact.aspx	78.46.71.246	4,966	200
28.01.2011 15:37:03	/Sverige/Partners/KnowIT.aspx	67.195.37.153	4,952	200
27.01.2011 10:35:45	/en/Partners/Become-Partner.aspx	65.61.164.180	4,952	200

After identifying URIs that either result from testing efforts or possible network issues due to geo-location, record the remaining.

If there is a wide spread number of URIs that are above the threshold, in our case 4 seconds, that occur during peak operation could be an indicator of a web site that does not have enough capacity to handle the load

2.2.6 Sitecore Recommendation

Sitecore recommends that the round trip time for any given .aspx page be under 4 seconds.

Report Findings

Record the Results

The peak hours of usage for the web site are: _____

Record any URIs that have a round trip time of greater than 4 seconds. These URLs are used during the Analyze Rendering Components task.

Note
The results from Log Parser Lizard GUI can be exported to Excel for cleanup and pasting into this document.

Timestamp	URI	Client IP	Status Code	Time

There are requests have exceeded the 4 second threshold __ YES __ NO

There is a large number of different URLs during peak operation hours __ YES __ NO.

There are requests that have exceeded the 4 second threshold = NO

OK. There are no request round trips that exceed 4 seconds.

There are requests that have exceeded the 4 second threshold = YES

Error. There are request round trips that have exceeded 4 seconds. Further investigation is required. Refer to the Rendering Component Performance Task.

There is a large number of different URLs during peak usage hours = YES

Error. There is a large number of different URIs that have exceed the 4 second threshold during peak usage hours. This could be an indication of capacity problems and requires further investigation.

2.3 Rendering Performance

By using the Sitecore stats page, information about rendering can be collected.

The stats page provides the following information about the various rendering used in a page, or throughout the site (depending when the stats page is observed).

- Rendering — The name of the rendering.
- Site — The name of the site that the information for the rendering is being collected.
- Count — The number of times that the rendering has been called since the last time the stats page was reset.
- From cache — The number of times the rendering was pulled from cache.
- Avg. time (ms) — The average time taken to render to output.
- Avg. items — The average number of items included in the rendering.
- Max. time — The maximum amount of time taken to render the output.
- Max. items — The maximum number of items included in the rendering.
- Total time — The total amount of time taken for all instances of this rendering since the last stats page reset occurred.
- Total items — The total number of items included in all instances of this rendering since the last stats page reset occurred.
- Last run — This the last time that stats were collected.

2.3.1 Required Skills

- A working knowledge of the Sitecore stats.aspx page.

2.3.2 Symptoms

- Slow webpage round trip times.

2.3.3 Procedure to Use the Sitecore Stats Page

This procedure requires that you have permissions to access aspx pages in the /sitecore/admin folder. Also, we recommended that the stats page be opened up in a separate tab or browser window, so that you can keep it open while using another tab or window to navigate the site you are investigating.

To launch the stats page:

1. Launch either two web browser windows or tabs. We refer to these as the **stats** window and the **site** window.
2. In the **stats** window, go to `http://<site>/sitecore/admin/stats.aspx`.
3. The **stats** window shows information about the renderings that have been requested since the last time the stats page was reset. If you want to start from scratch, click the refresh button (for example, if you want to view information regarding an individual web page you must clear out the stats page).
4. If you wish to view rendering stats that have been collected since the last reset, use the information presented.
5. If you wish to collect information about a single page, reset the stats page. In the **site** window, go to the website or webpage that you are going to collect stats about the renderings it

includes. Note: make several calls to the page so that information about caching and averages can be collected.

6. We recommended that you export the information in the stats page table to Excel, so that the information can be sorted for easier data analysis. To export, right click the table and click **Export to Microsoft Excel**.

2.3.4 Understanding the Results

The following stats table has been exported into Excel, sorted by Max. time, and has some of the columns hidden.

	A	B	C	D	E	G
1	Rendering	Site	Count	From cache	Avg. time (ms)	Max. time
2	Placeholder: content	nicam	20	0	185.3304	1670.1252
3	Sublayout: /layouts/Nicam/HomePageContent.ascx	nicam	2	0	1080.6819	1669.1879
4	Placeholder: rightcolumn	nicam	15	0	142.9569	1437.7889
5	/xsl/Nicam/NewsSpot.xslt	nicam	2	0	938.391	1419.1291
6	Sublayout: /layouts/Nicam/ThreeColumnContent.ascx	nicam	13	0	86.9166	362.6216
7	Placeholder: centercolumn	nicam	18	0	50.8885	328.2586
8	Sublayout: /layouts/Nicam/ProductForums.ascx	nicam	1	0	327.9864	327.9864
9	Sublayout: /layouts/Nicam/TwoColumnContent.ascx	nicam	5	0	82.8207	274.0745
10	Sublayout: /layouts/Nicam/ContactUsFormWrapper.ascx	nicam	1	0	260.099	260.099
11	FormRender1 (FormRender)	nicam	1	0	260.0562	260.0562
12	Placeholder: forum-content	nicam	1	0	178.4444	178.4444
13	Sublayout: /sitecore modules/Web/YAF/YAF_Forum.ascx	nicam	1	0	178.4045	178.4045
14	Placeholder: phxml	nicam	2	0	86.4851	162.7259
15	/xsl/FlashImageIterator/Flash_XMLOutput.xslt	nicam	2	0	86.4505	162.6944
16	/xsl/Nicam/Logo.xslt	nicam	21	7	12.5506	146.9511
17	Placeholder: frontpagebottomspotbar	nicam	2	0	84.1091	142.2216
18	Sublayout: /layouts/Nicam/Spots Three Column.ascx	nicam	4	0	42.0658	142.2035
19	Placeholder: spotbarcenter	nicam	4	0	33.869	121.6814
20	/xsl/Nicam/RotateSpots.xslt	nicam	6	0	32.8522	121.6473
21	/xsl/Nicam/FlexiblePersonalizationSpot.xslt	nicam	9	0	13.5927	111.2867
22	/xsl/Nicam/Top Menu.xslt	nicam	21	0	17.2713	106.0543
23	/xsl/Nicam/Product Catalog.xslt	nicam	12	0	17.6432	100.0303
24	Placeholder: frontflash	nicam	2	0	45.6854	89.0685

The following observations can be made from this table:

- All renderings that have *Max time* in excess of 100ms need to be investigated to see if recommended coding practices have been followed.
- 0's in the *From cache* column indicate that Rendering (HTML Output) caching has not been configured.

2.3.5 Sitecore Recommendation

We recommend that renderings have a *Max time* of < 100ms, and that Rendering (HTML Output) caching be enabled and configured.

Report Findings

Record the Results

Rendering *Max times* are < 100ms: __YES __NO

There is several *From cache* values equal to 0: __YES __NO

Rendering *Max times* are < 100ms = YES:

OK. No rendering has a *Max time* > 100ms, per Sitecore recommended practices.

Rendering *Max times* are < 100ms = NO:

Error. There are renderings with a *Max time* > 100ms. Sitecore recommended practices are to keep rendering *Max times* less than 100ms.

There are several *From cache* values equal to 0 = NO:

OK. Rendering (HTML Output) caching has been enabled and configured per Sitecore recommended practices.

There are several *From cache* values equal to 0 = YES:

Error. Rendering (HTML Output) caching is either not enabled, and/or not configured. Sitecore recommended practices are to enable and configure Rendering (HTML Output) caching to improve site performance. For more information on how to enable and configure the Rendering caching, see the *Cache Configuration Reference* and the *Presentation Component Reference* manuals

2.4 Investigating Memory Leaks with the Sitecore Logs

The Sitecore system provides a series of performance counters that are logged to the Sitecore log file(s) on a ten-minute interval:

- Process\Private Bytes
- Process\Virtual Bytes
- Process\Page File Bytes
- .net CLR Memory\# Bytes in all Heaps
- .net CLR Memory\% Time in GC
- .net CLR Memory\Large Object Heap size
- .net CLR Loading\Bytes in Loader Heap
- .net CLR Loading\Current Assemblies

If the Sitecore performance counters are not available, they can be downloaded from:

http://sdn.sitecore.net/upload/sdn5/faq/administration/sitecorecounters.zip.

This article explains possible problems with the counters and solutions for them:
http://sdn.sitecore.net/Scrapbook/Working%20with%20Sitecore%20Performance%20Counters.aspx

The two counters that are of interest for this task are the *Process\Private Bytes* and the *.net CLR Memory\# Bytes in all Heaps.*

The *Process\Private Bytes* counter reports all memory that is exclusively allocated for the process (w3wp.exe) and can't be shared with other processes on the system. And the *.net CLR Memory\# Bytes in all Heaps* counter reports the combined total size of the Gen0, Gen1, Gen2, and large object heaps.

Typically the Private Bytes and # Bytes in all Heaps rise and fall at the same rate. If the Private Bytes is increasing, but the # Bytes in all Heaps remains stable, unmanaged memory is leaking. If both are increasing, and not being cleared, then there is a potential for a leak in the managed memory.

By using the Sitecore log files, *Log Parser Lizard GUI* and *Excel*, we graph these two counters to look for potential leaks.

2.4.1 Required Skills

- A working knowledge of the Sitecore Logs.
- A working knowledge of SQL scripting.
- A working knowledge of Log Parser Lizard GUI.
- A working knowledge of graphing with Microsoft Excel.

2.4.2 Symptoms

- OutOfMemory Exceptions
- IIS App Pool recycling
- Sluggish performance as memory usage increases.

2.4.3 Parsing the Sitecore Log(s) using Log Parser Lizard GUI

Information about installing and setting up the Log Parser software can be found at http://www.theclientview.net/?p=51

This task requires querying for two different result sets. The first is to query for all Process\Private Bytes Messages, and parsing out the values. The second is to query for all .net CLR Memory\# Bytes in all Heaps, parsing out the values.

The results are copied into Excel so that they can be graphed and compared.

Query for all Process\Private Bytes

This query parses all the log entries for Message fields that contain Health.Counter and Process\Private Bytes. The information is presented with the log file name, the time of the log entry, the name of the performance counter, as well as the value (in bytes).

DateTime	Counter	Bytes
22.12.2010 09:48:21	Private Bytes	776,396,800
22.12.2010 09:58:26	Private Bytes	825,212,928
22.12.2010 10:08:31	Private Bytes	848,396,288
22.12.2010 10:18:36	Private Bytes	963,715,072
22.12.2010 10:28:41	Private Bytes	979,030,016
22.12.2010 10:38:46	Private Bytes	928,858,112
22.12.2010 10:48:50	Private Bytes	963,055,616
22.12.2010 10:58:55	Private Bytes	1,073,197,056
22.12.2010 11:09:00	Private Bytes	1,151,533,056
22.12.2010 11:19:05	Private Bytes	1,341,677,568
22.12.2010 11:29:10	Private Bytes	1,182,515,200

1. Launch **Log Parser Lizard GUI**.
2. Create a **New Query**.
3. Change the Input Log Format to **Multiline Regex (Log4Net)** Input Format.
4. Run the following script (change the `#logs location#, for example "c:\logs\*.txt"`):

```
SELECT DateTime, Counter, TO_INT(Value) AS Bytes
        USING
        /* Parse the date from the Filename field */
        INDEX OF(Filename, 'log') AS startDate,
        SUBSTR(Filename, ADD(startDate,4), 8) AS tempDate,

        /* convert date to MM/DD/YYYY format */
        SUBSTR(tempDate, 0, 4) AS Year,
        SUBSTR(tempDate, 4, 2) AS Month,
        SUBSTR(tempDate, 6, 2) AS Day,
        STRCAT(Month, '/') AS datePart1,
        STRCAT(datePart1, Day) AS datePart2,
        STRCAT(datePart2, '/') AS datePart3,
        STRCAT(datePart3, Year) AS strDate,

        /* create date time string */
        STRCAT(strDate, ' ') AS strDateTimePart1,
        STRCAT(strDateTimePart1, Time) AS strDateTime,

        /* Create TimeDate Stamp */
        TO_TIMESTAMP(strDateTime, 'MM/dd/yyyy HH:mm:ss') AS DateTime,

        /* Parse the name of the counter from the Message field */
        LAST_INDEX_OF(Message, ')') AS endCounter,
```

```
        INDEX_OF(Message, '\\') AS startCounter,
        SUBSTR(Message, ADD(startCounter,1), SUB(SUB(endCounter,startCounter),1)) AS Counter,

        /* Parse the number of bytes from the Message field */
        LAST_INDEX_OF(Message, ' ') as startRawValue,
        EXTRACT_TOKEN(SUBSTR(Message, startRawValue), 1, ' ') as rawValue,
        REPLACE_CHR(rawValue, ',', '') AS Value

        FROM #logs location#
        WHERE Message LIKE '%health.counter%' AND Message LIKE '%private bytes%'
```

5. In the **Tools** menu, click **Export grid to Excel**. Save with a meaningful name and open the xsl file.

Query for all # Bytes in all Heaps

This query parses all log entries for Message fields that contain Health.Counter and .net CLR Memory\# Bytes in all Heaps. The information is presented with the log file name, the time of the log entry, the name of the performance counter, as well as the value (in bytes).

DateTime	Counter	Bytes
22.12.2010 09:48:21	# Bytes in all Heaps	471,164,984
22.12.2010 09:58:26	# Bytes in all Heaps	588,572,200
22.12.2010 10:08:31	# Bytes in all Heaps	646,315,600
22.12.2010 10:18:36	# Bytes in all Heaps	744,769,840
22.12.2010 10:28:41	# Bytes in all Heaps	673,051,952
22.12.2010 10:38:46	# Bytes in all Heaps	710,841,824
22.12.2010 10:48:50	# Bytes in all Heaps	743,166,456
22.12.2010 10:58:55	# Bytes in all Heaps	843,086,048
22.12.2010 11:09:00	# Bytes in all Heaps	915,655,080
22.12.2010 11:19:05	# Bytes in all Heaps	1,070,248,360
22.12.2010 11:29:10	# Bytes in all Heaps	924,331,240
22.12.2010 11:39:15	# Bytes in all Heaps	962,719,544

1. Launch **Log Parser Lizard GUI**.
2. Create a **New Query**.
3. Change the Input Log Format to **Multiline Regex (Log4Net)** Input Format.
4. Run the following script (change the `#logs location##, for example "c:\logs\*.txt"`):

```
SELECT DateTime, Counter, TO_INT(Value) AS Bytes
        USING
        /* Parse the date from the Filename field */
        INDEX_OF(Filename, 'log') AS startDate,
        SUBSTR(Filename, ADD(startDate,4), 8) AS tempDate,

        /* convert date to MM/DD/YYYY format */
        SUBSTR(tempDate, 0, 4) AS Year,
        SUBSTR(tempDate, 4, 2) AS Month,
        SUBSTR(tempDate, 6, 2) AS Day,
        STRCAT(Month, '/') AS datePart1,
        STRCAT(datePart1, Day) AS datePart2,
        STRCAT(datePart2, '/') AS datePart3,
        STRCAT(datePart3, Year) AS strDate,

        /* create date time string */
        STRCAT(strDate, ' ') AS strDateTimePart1,
        STRCAT(strDateTimePart1, Time) AS strDateTime,

        /* Create TimeDate Stamp */
```

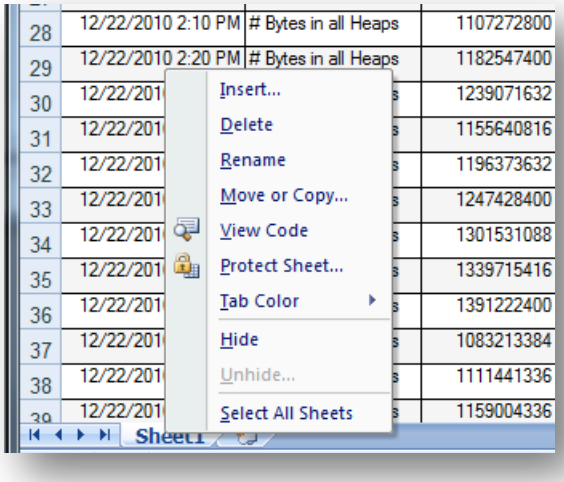
```
        TO_TIMESTAMP(strDateTime, 'MM/dd/yyyy HH:mm:ss') AS DateTime,

        /* Parse the name of the counter from the Message field */
        LAST_INDEX_OF(Message, ')') AS endCounter,
        INDEX_OF(Message, '\\') AS startCounter,
        SUBSTR(Message, ADD(startCounter,1), SUB(SUB(endCounter,startCounter),1)) AS Counter,

        /* Parse the number of bytes from the Message field */
        LAST_INDEX_OF(Message, ' ') as startRawValue,
        EXTRACT_TOKEN(SUBSTR(Message, startRawValue), 1, ' ') as rawValue,
        REPLACE_CHR(rawValue, ',', '') AS Value

        FROM #logs location#
        WHERE Message LIKE '%health.counter%' AND Message LIKE '%bytes in all
heaps%'
```

5. In the **Tools** menu, click **Export grid to Excel**. Save as **temp.xls** and open the xsl file.

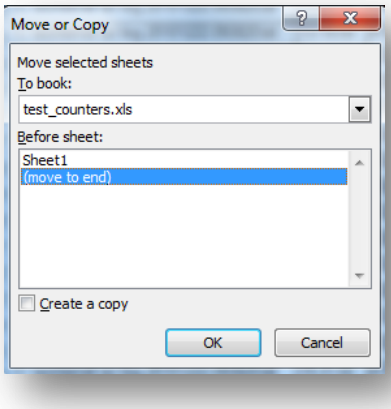
2.4.4 Graphing the Results

There are several ways to create graphs in Excel. The steps below are for creating a line graph to compare the Process\Private Bytes results set to the .net CLR Memory\# Bytes in all Heaps.

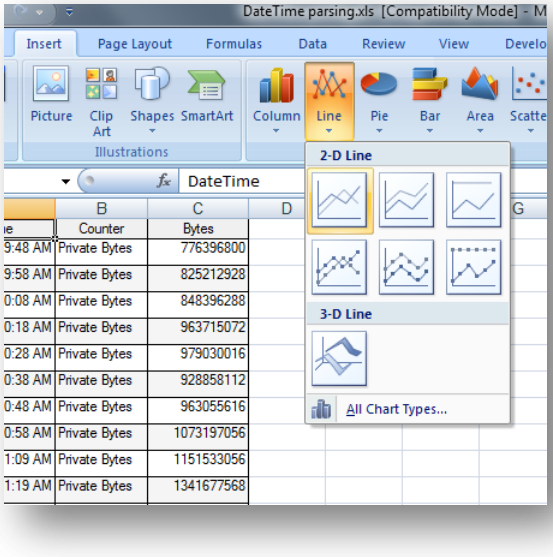
1. The first step is to combine both spread sheets into one workbook. Both spreadsheets that were created above need to be open.
2. Switch to the `temp.xls` spreadsheet created above.
3. Right mouse click the **Sheet 1** tab (located at the bottom of the spreadsheet) and click **Move or Copy...**



4. Move the Sheet to the first Excel document that was created.

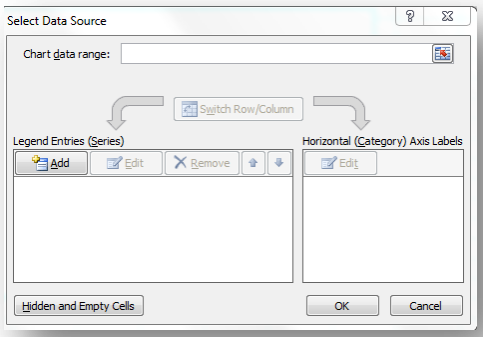


5. Close `temp.xls`.
6. Switch to the first spreadsheet created above, it should now contain two sheets. One with the results set from the *Process\Private Bytes* query and the other from the .net *CLR Memory\# Bytes in all Heaps* query.
7. If your data contains information from more than one log file, be sure to sort (via the **Data** tab) both sheets by the **DateTime** column. Otherwise the results in the graph appear strange.
8. From the **Insert** tab click **Charts / Line / 2D Line**

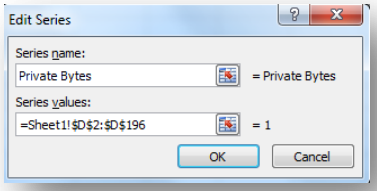


9. In the **Data** group, click **Select Data**. If something was selected on the sheet, it attempts to graph it. Clear the graph by removing any series and clear the Chart data range. You are

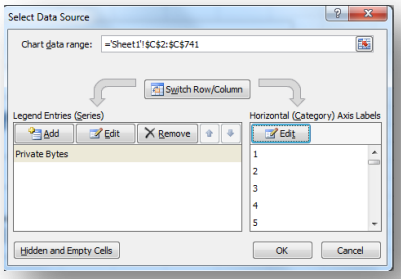
presented with the **Select Data Source** dialog box.

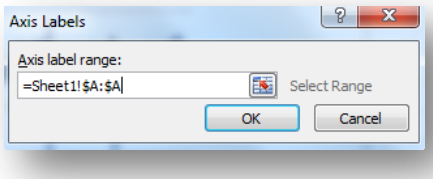


10. Click **Add**.
11. For the first series, name the series **Private Bytes** and select the data range from the Bytes column.

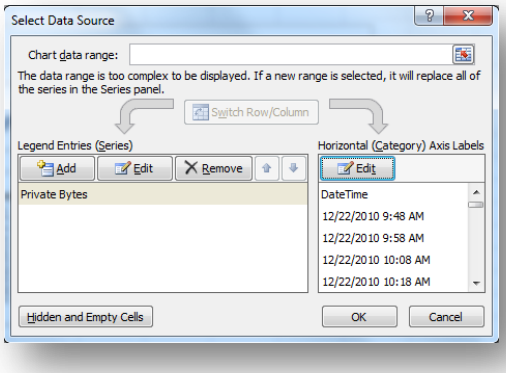


12. Next, set the Horizontal (x-axis) labels to the **DateTime** column (column A), click **Edit** and selecting column A from sheet 1:

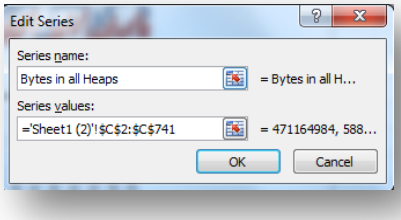


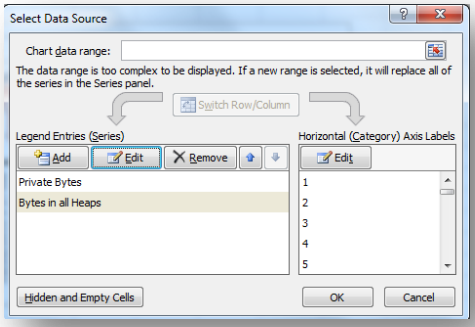


13. Click **OK**.



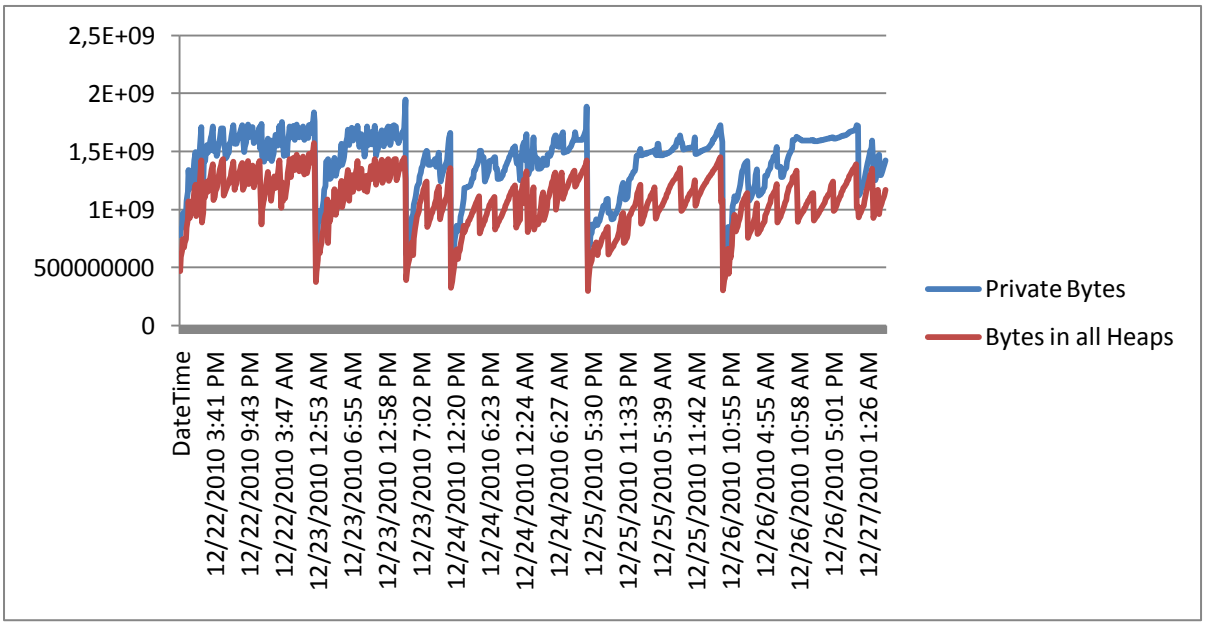
14. Click **Add** and do the same for the second series (Bytes in all Heaps), selecting the data range from the bytes column on the sheet containing the Bytes in all Heaps results set. Also note that you need to set the Horizontal (x-axis) for the Bytes in all Heaps to Column A of sheet 1.





15. Click **OK**.

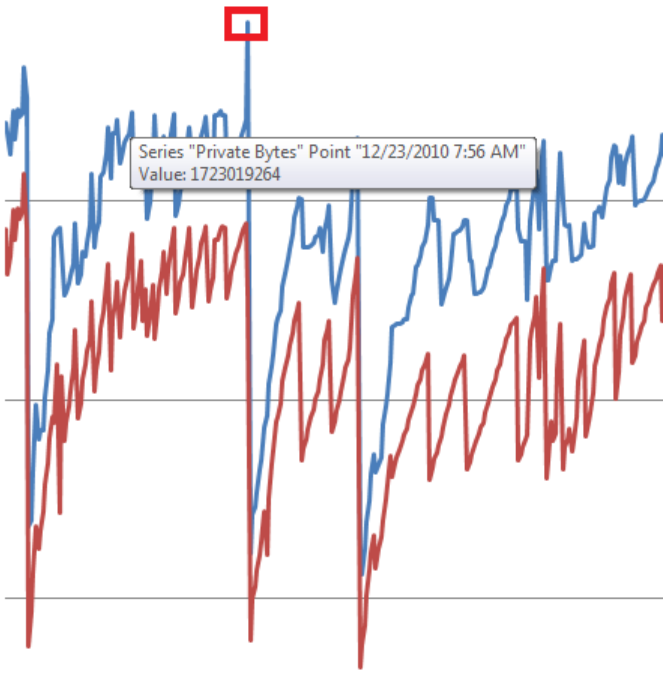
2.4.5 Understanding the Results (Graph)



Graph 1

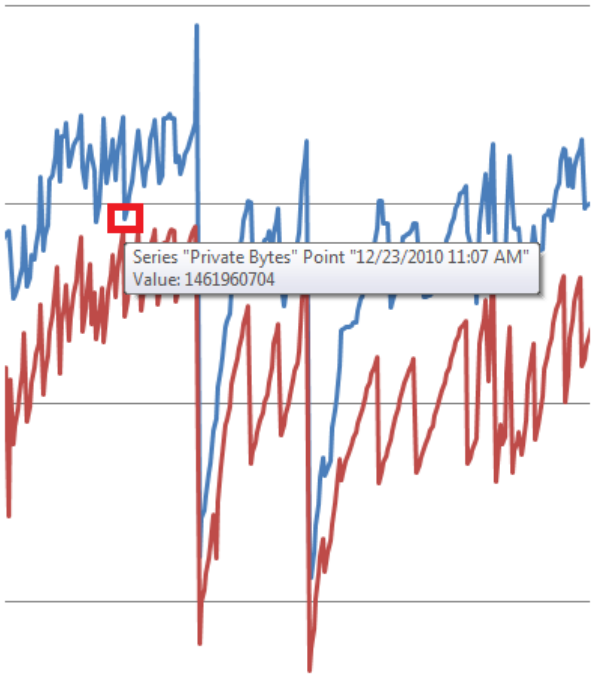
- This graph illustrates an example of a memory leak in the managed memory space. The five big dips represent app pool restarts due to either the app pool private memory limit being reached, or OOM exceptions. Between each dip you can see Private bytes and Heap bytes trending an increase in parallel. This is the symptom of a memory leak.
- Looking at our sample, we can see that both the Private Bytes and the Bytes in all Heaps are rising and falling at the same rate. It is OK for memory to rise to a stable point and remain there. Note however that our sample graph shows 5 events where memory is reset. Further investigation is explained later.
- If Private Bytes were rising, while Bytes in all Heaps were to remain constant, this would be an indication that there is a potential leak in the unmanaged memory space and further investigation would be required.

- Peaks indicate memory allocation. Moving the mouse over a peak provides information about the amount of memory allocated, as well as the time of the event:



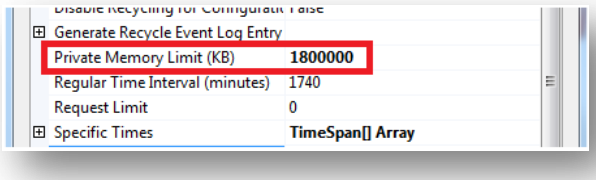
Graph 2

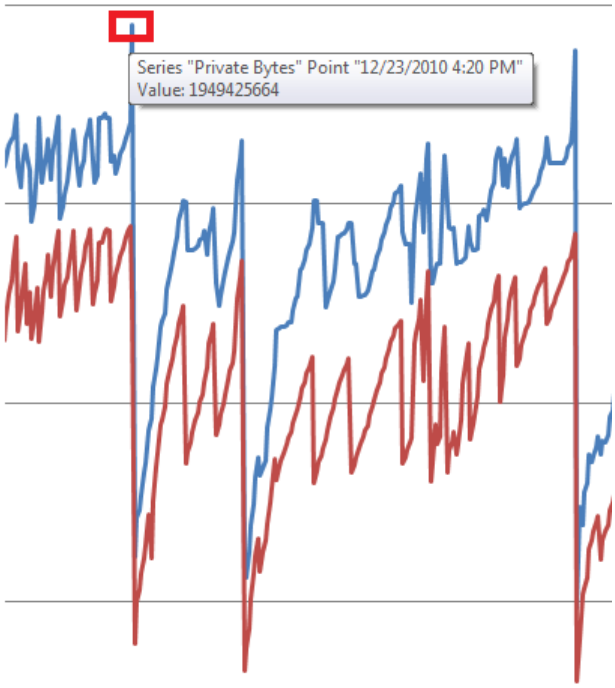
- Detail view of Private Bytes spike, indicating peak memory usage prior to an app pool recycle or OOM exception.
- Valleys indicate when a garbage collection event has occurred. Moving the mouse over a valley provides information as to the memory still allocated, as well as the time that the event occurred:



Graph 3

- Details of a valley or GC event. Note, this is not an app pool recycle or OOM exception, but the normal activity of the garbage collector.
- By looking at the highest peaks and the lowest valleys, we can use this information to correlate information that we can obtain from the Windows event logs.
- The high points, just prior to the memory being reset can be compared to the private memory limit set for the App Pool (Go to the IIS Manager, select the appropriate App Pool, and select Advanced Settings). Please note: If there is no value set for the Private Memory limit on the App Pool, check the Sitecore logs for OutOfMemory Exceptions occurring during the same time frame. (Below, we can see that the App Pool is set to reset when memory reaches 1800000KB).

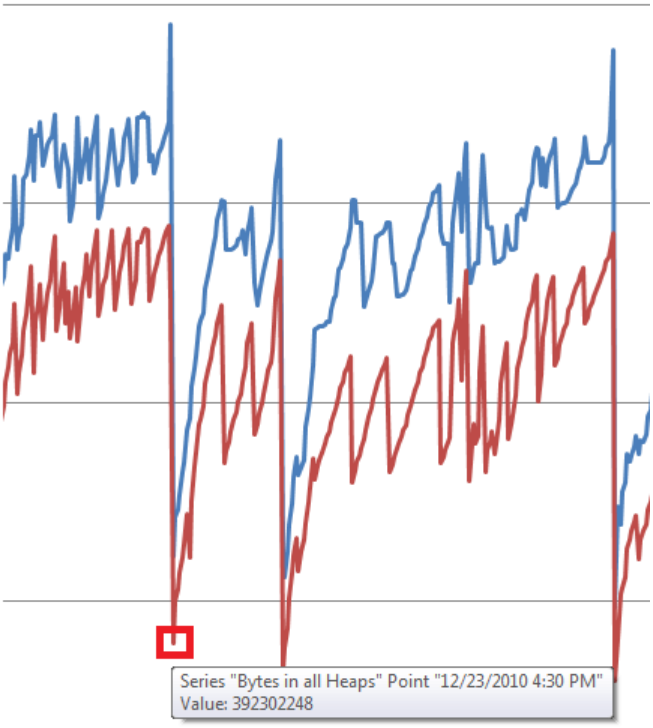




Graph 4

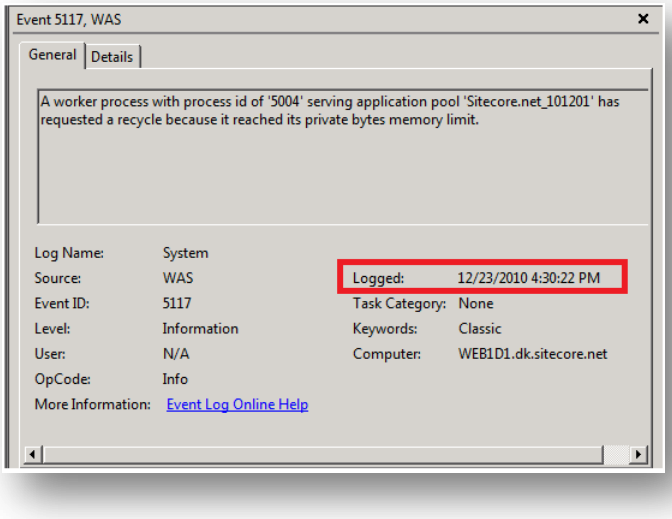
- Details of a memory spike just prior to an app pool recycle or OOM exception. This information can be used to determine if the memory usage exceeds that of the available memory for the app pool.
- By comparing the low point that follows the peak with information from the Windows event logs, we can see that an App Pool recycle did indeed take place. This constant increase in both Private Bytes and Bytes in all Heaps, followed by exceeding the memory limits set and a recycling of the App Pool could be a potential leak in the managed memory space. Further

investigation would be required.



Graph 5

- Details of a memory reset event. The date and time available can be used to correlate the information available from the Sitecore Logs and the Windows Event Logs to see if an OOM exception and / or an app pool recycling event has taken place.



(The time stamps match, indicating that the App Pool recycled.)

2.4.6 Notes

Additional information about .net memory usage and leak investigation can be found at:

- http://msdn.microsoft.com/en-us/magazine/cc163491.aspx

- http://msdn.microsoft.com/en-us/library/Ee817660(pandp.10).aspx

Chapter 3

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
- Check for SQL Server Maintenance Plan
- Cleanup Database Tables
- Check Database Cleanup Agents
- Disable Search Indexes if Not Used.
- Software and Server Configuration

3.1 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*.

3.1.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.1.2 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.

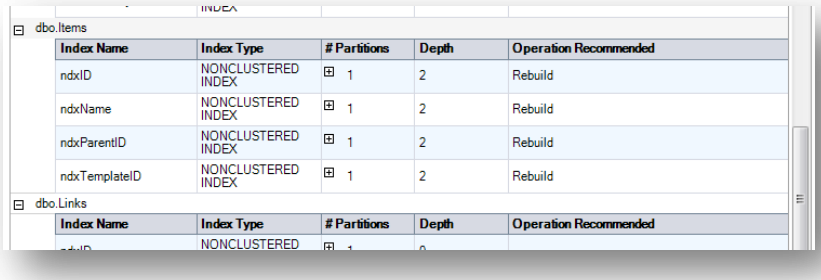
3.1.3 Procedure to Check for Fragmented Indexes

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.

3.1.4 Understanding the Results

The output looks like this:



dbo.Items

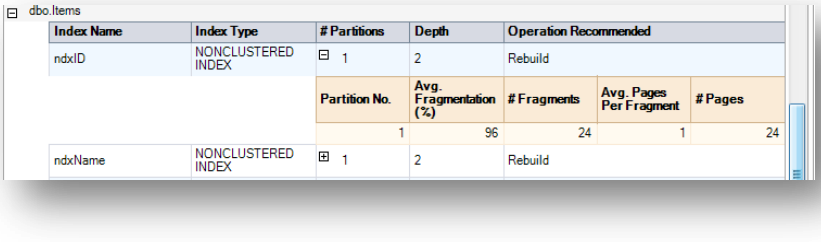
Index Name	Index Type	# Partitions	Depth	Operation Recommended
ndxID	NONCLUSTERED INDEX	1	2	Rebuild
ndxName	NONCLUSTERED INDEX	1	2	Rebuild
ndxParentID	NONCLUSTERED INDEX	1	2	Rebuild
ndxTemplateID	NONCLUSTERED INDEX	1	2	Rebuild

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
		Partition No.	Avg. Fragmentation (%)	# Fragments	Avg. Pages Per Fragment	# Pages
		1	96	24	1	24
ndxName	NONCLUSTERED INDEX	1	2	Rebuild		

3.1.5 Sitecore Recommendation

Sitecore recommends keeping index fragmentation below 10%.

3.1.6 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 *Check for SQL Server Maintenance Plan.*

Report Findings

Record the Results

Are any of the indexes fragmented > 10% __YES __NO

Page fragmentation is < 10%:

OK. The results from the index fragmentation inspection shows that your indexes on the CMS databases are not fragmented to an extent that affects performance

Page fragmentation is > 10%:

Error. The results from the index fragmentation inspection shows that your indexes on the CMS databases are fragmented. It is Sitecore's recommendation that first the database(s) are backed up, and then defragmented.

3.2 Check for 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.

3.2.1 Required Skills

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

3.2.2 Symptoms

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

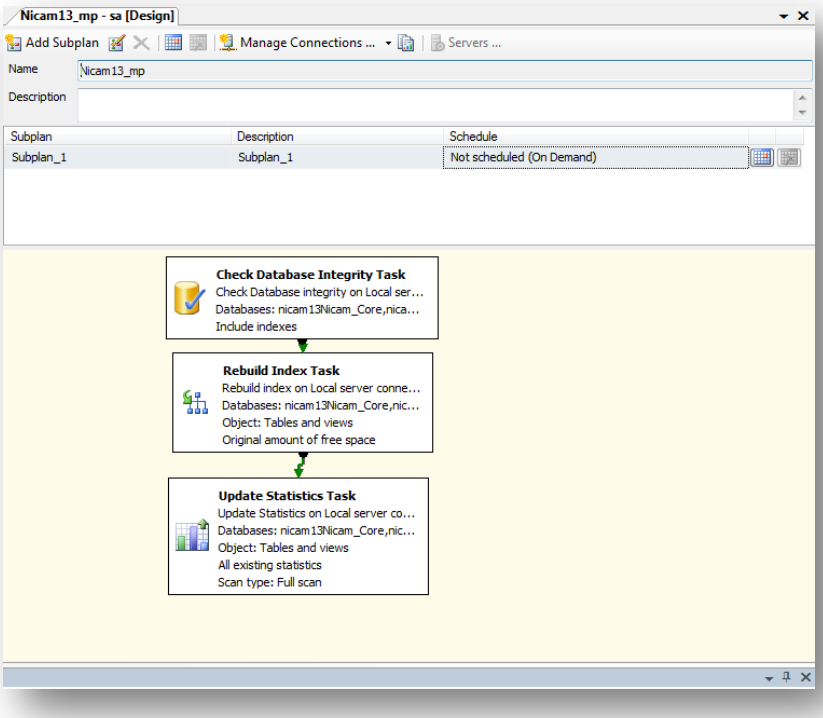
3.2.3 Procedure to Check for SQL Server Maintenance Plan

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).

3.2.4 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.

3.2.5 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.

3.2.6 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.*
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), 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), 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**.

Report Findings

Record the Results

Does a Maintenance Plan exist? __ YES __NO

Is the Maintenance Plan setup against the Analytics database? __YES __NO

Is the MP scheduled or on demand? __ SCH __On Demand

If scheduled, what is the interval? _____

What steps does the MP perform?

Maintenance Plan is in place and follows best practices as outlined in the "Findings" below:

- Does a Maintenance Plan exist? **YES**
- Is the Maintenance Plan setup against the Analytics database? **YES**
- Is the MP scheduled or on demand? — **SCH**
- If scheduled, what is the interval? — **Daily**
- What steps does the MP perform? — Check Database Integrity, Rebuild Index Task, Update Statistics

Ok. The results of the SQL Server Maintenance Plan check shows that a Maintenance Plan is in place, properly scheduled and follows Sitecore Best Practices

Maintenance Plan is in place, but does not follow best practices:

Error. The results of the SQL Server Maintenance Plan check shows that a Maintenance Plan is in place, but does not follow Sitecore best practices.

Maintenance Plan does not exist:

Error. The results of the SQL Server Maintenance Plan check shows that a Maintenance Plan currently does not exist

3.2.7 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.

3.3 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.

3.3.1 Required Skills

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

3.3.2 Symptoms

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

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

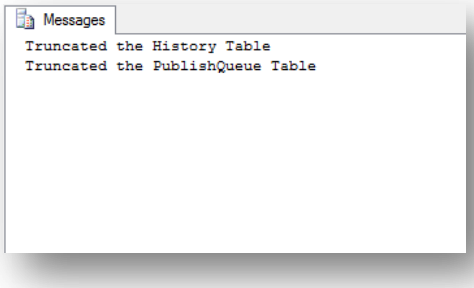
To clean up *the History, PublishQueue, and EventQueue* tables:

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
```

3.3.4 Understanding the Results

The output looks similar to this:



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

3.3.5 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.

3.3.6 How to solve

Run the procedure described earlier.

Report Findings

Record the Results

The *History*, *PublishQueue*, or *EventQueue* table needed to be truncated: Yes__ No__

The *History*, *PublishQueue*, or *EventQueue* table needed to be truncated: = No

OK. The number of rows in the *History*, *PublishQueue*, or *EventQueue* table is less than 1000, per Sitecore recommended practices.

The *History*, *PublishQueue*, or *EventQueue* table needed to be truncated: = Yes

Error. The number of rows in the *History*, *PublishQueue*, or *EventQueue* table is greater than 1000, Sitecore recommends that the number of rows (entries) in the *History*, *PublishQueue*, and *EventQueue* tables be less than 1000.

3.4 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.

3.4.1 Required Skills

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

3.4.2 Symptoms

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

3.4.3 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 on a periodic base:

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

3.4.4 Understanding the Results

The output looks like this (some <agent> nodes 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**.

3.4.5 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**.)

3.4.6 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)

Report Findings

Record the Results

Scheduling frequency is greater than 00:00:00 Yes__ No__

Sitecore.Tasks.CleanupHistory interval is greater than 00:00:00 Yes__ No__

Sitecore.Tasks.CleanupPublishQueue interval is greater than 00:00:00 Yes__ No__

Sitecore.Tasks.CleanupEventQueue interval is greater than 00:00:00 Yes__ No__

Scheduling frequency is greater than 00:00:00 = Yes:

OK. The scheduling frequency is set to greater than 00:00:00, allowing the agents to run at the set intervals.

Scheduling frequency is greater than 00:00:00 = No:

Error. The scheduling frequency is set to 00:00:00, disallowing the agents to run at their set intervals. Sitecore recommends that the scheduling frequency be set to greater than 00:00:00.

Sitecore.Tasks.CleanupHistory interval is greater than 00:00:00 = Yes:

OK. The Sitecore.Tasks.CleanupHistory agent's interval is set to greater than 00:00:00, allowing it to run on a periodic basis.

Sitecore.Tasks.CleanupHistory interval is greater than 00:00:00 = No:

Error. The interval for Sitecore.Tasks.CleanupHistory is set to 00:00:00, disallowing the agent to run at a periodic basis. Sitecore recommends that the interval be set to greater than 00:00:00.

Sitecore.Tasks.CleanupPublishQueue interval is greater than 00:00:00 = Yes:

OK. The Sitecore.Tasks.CleanupPublishQueue agent’s interval is set to greater than 00:00:00, allowing it to run on a periodic basis.

Sitecore.Tasks.CleanupPublishQueue interval is greater than 00:00:00 = No:

Error. The interval for Sitecore.Tasks.CleanupPublishQueue is set to 00:00:00, disallowing the agent to run at a periodic basis. Sitecore recommends that the interval be set to greater than 00:00:00.

Sitecore.Tasks.CleanupEventQueue interval is greater than 00:00:00 = Yes:

OK The Sitecore.Tasks.CleanupEventQueue agent’s interval is set to greater than 00:00:00, allowing

it to run on a periodic basis.

Sitecore.Tasks.CleanupEventQueue interval is greater than 00:00:00 = No:

Error. The interval for Sitecore.Tasks.CleanupEventQueue is set to 00:00:00, disallowing the agent to run at a periodic basis. Sitecore recommends that the interval be set to greater than 00:00:00.

3.5 Disable Search Indexes if Not Used.

The processes of rebuilding and / or updating indexes can be resource intensive. If an search indexes are not being used, they should not be updated.

This task describes how to disable index updating.

3.5.1 Required Skills

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

3.5.2 Symptoms

- Slow performance during indexing.

3.5.3 Procedure to Disable Search Index Updates

The following procedure describes where to look to see if the cleanup tasks are configured to run on a periodic base:

1. Open up the `web.config` file in your favorite editor.
2. Set the update interval time for Indexing.UpdateInterval to 00:00:00 `<setting name="Indexing.UpdateInterval" value="00:00:00" />`

3.5.4 Sitecore Recommendation

Sitecore recommends disabling indexing updating if search indexes are not being used by setting the `Indexing.UpdateInterval` to a value of 00:00:00.

3.5.5 How to solve

See the section *Procedure to Disable Search Index Updates*.

Report Findings

Record the Results

Search Indexing is being used: __ Yes __ No

Index.UpdateInterval value = _____

Search Indexing is being used = Yes:

Search indexing is in use, this task does not apply.

Search indexing is being used = No **AND** Indexing.UpdateInterval = 00:00:00

OK. Search indexing is not being used and the update interval = 00:00:00, per Sitecore recommended practices.

Search indexing is being used = No **AND** Indexing.UpdateInterval > 00:00:00

Error. Search indexing is not being used and the update interval > 00:00:00, Sitecore recommends disabling indexing updating if search indexes are not being used by setting the Indexing.UpdateInterval value to 00:00:00

3.6 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.

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
- 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 Production Environment (Web Server):

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

Sitecore recommends the following for the Production 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 Sitecore Analytics (OMS 1.x) (SQL Server):

- SQL Server 2008 R2 x64

- Dual Quad Core processors
- 16 GB RAM
- 500 GB disk for data files
- 250 GB disk for log files
- 250 GB disk for temp space

3.6.1 Required Skills

- A working knowledge of system infrastructures.

3.6.2 Symptoms

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

3.6.3 Server Configuration Scoring

Scoring the server, software and hardware, configuration is broken down into scoring tables. The score that is recorded in each table determines whether or not the system meets, exceeds or falls short of the Sitecore recommendations.

General Requirements / OS All Servers

Windows Server	2008 R2 x64	2008 x64	2003 x64	32 bit OS
	1	-1	-2	-4

Score _____

Content Authoring Environment (Web Server)

IIS Version	7.x	6.x	5.x
	+1	-1	-4

Score _____

Processor Cores	8+	4	2	1
	+2	1	-1	-2

Score_____

RAM	16 GB +	12 GB	8 GB	4 GB	2 GB -
	+2	+1	1	-2	-4

Score_____

Content Authoring Environment (SQL Server)

SQL Server Version	2008 R2	2008	2005
	1	-1	-2

Score _____

Processor Cores	12+	8	4	2
	+2	1	-1	-2

Score_____

RAM	16 GB +	12 GB	8 GB	4 GB	2 GB -
	+2	1	-1	-2	-4

Score_____

Production Environment (Web Server)

IIS Version	7.x	6.x	5.x
	+1	-1	-4

Score _____

Processor Cores	12+	8	4	2
	+2	1	-1	-2

Score_____

RAM	16 GB +	12 GB	8 GB	4 GB	2 GB -
	+2	+1	1	-2	-4

Score_____

Production Environment (SQL Server)

SQL Server Version	2008 R2	2008	2005
	1	-1	-2

Score _____

Processor Cores	12+	8	4	2
	+2	1	-1	-2

Score_____

RAM	16 GB +	12 GB	8 GB	4 GB	2 GB -
	+2	1	-1	-2	-4

Score_____

Sitecore Analytics (OMS 1.x) (SQL Server)

SQL Server Version	2008 R2	2008	2005
	1	-1	-2

Score _____

Processor Cores	12+	8	4	2
	+2	1	-1	-2

Score_____

RAM	20 GB +	16 GB	12 GB	8 GB	4 GB -
	+2	1	-1	-2	-4

Score_____

Separate disks for data, logs, temp	Yes	No
	1	-2

Score_____

Report Findings

General requirements / OS All Servers

Score >= 1

OK. The installed OS version meets or exceeds the Sitecore recommended requirements.

Score < 1

Error. The installed OS version does not meet the Sitecore recommended. Sitecore highly recommends Microsoft Windows 2008 R2 for it's enhanced features and resolved bug issues.

Content Authoring Environment (Web Server) – IIS Version

Score >= 1

OK. The IIS version running meets or exceeds the Sitecore recommended requirements.

Score < 1

Error. The IIS version does not meet the Sitecore recommended requirements. Sitecore highly recommends running IIS version 7.x.

Content Authoring Environment (Web Server) – Processor Cores

Score >= 1

OK. The number of CPU cores installed on the server meets or exceeds the Sitecore recommended requirements of 4 processor cores.

Score < 1

OK. The number of CPU cores installed on the server does not meet the Sitecore recommended requirements of 4 processor cores.

Content Authoring Environment (Web Server) – RAM

Score >= 1

OK. The amount of RAM installed on the server meets or exceeds the Sitecore recommended requirements 8 GB RAM.

Score < 1

OK. The amount of RAM installed on the server does not meet the Sitecore recommended requirements of 8 GB RAM.

Content Authoring Environment (SQL Server) – SQL Server Version

Score >= 1

OK. The SQL Server version running meets or exceeds the Sitecore recommended requirements.

Score < 1

Error. The SQL Server version does not meet the Sitecore recommended requirements. Sitecore highly recommends running SQL Server 2008 R2 x64.

Content Authoring Environment (SQL Server) – Processor Cores

Score >= 1

OK. The number of CPU cores installed on the server meets or exceeds the Sitecore recommended requirements 8 processor cores.

Score < 1

OK. The number of CPU cores installed on the server does not meet the Sitecore recommended requirements of 8 processor cores.

Content Authoring Environment (SQL Server) – RAM

Score >= 1

OK. The amount of RAM installed on the server meets or exceeds the Sitecore recommended requirements of 12 GB RAM.

Score < 1

OK. The amount of RAM installed on the server does not meet the Sitecore recommended requirements of 12 GB RAM.

Production Environment (Web Server) – IIS Version

Score >= 1

OK. The IIS version running meets or exceeds the Sitecore recommended requirements.

Score < 1

Error. The IIS version does not meet the Sitecore recommended requirements. Sitecore highly recommends running IIS version 7.x.

Production Environment (Web Server) – Processor Cores

Score >= 1

OK. The number of CPU cores installed on the server meets or exceeds the Sitecore recommended requirements of 8 processor cores.

Score < 1

OK. The number of CPU cores installed on the server does not meet the Sitecore recommended requirements of 8 processor cores.

Production Environment (Web Server) – RAM

Score >= 1

OK. The amount of RAM installed on the server meets or exceeds the Sitecore recommended requirements of 8 GB RAM

Score < 1

OK. The amount of RAM installed on the server does not meet the Sitecore recommended requirements of 8 GB RAM.

Production Environment (SQL Server) – SQL Server Version

Score >= 1

OK. The SQL Server version running meets or exceeds the Sitecore recommended requirements.

Score < 1

Error. The SQL Server version does not meet the Sitecore recommended requirements. Sitecore highly recommends running SQL Server 2008 R2 x64.

Production Environment (SQL Server) – Processor Cores

Score >= 1

OK. The number of CPU cores installed on the server meets or exceeds the Sitecore recommended requirements of 8 processor cores.

Score < 1

OK. The number of CPU cores installed on the server does not meet the Sitecore recommended requirements 8 processor cores.

Production Environment (SQL Server) – RAM

Score >= 1

OK. The amount of RAM installed on the server meets or exceeds the Sitecore recommended requirements of 12 GB RAM.

Score < 1

OK. The amount of RAM installed on the server does not meet the Sitecore recommended requirements of 12 GB RAM.

Sitecore Analytics (OMS 1.x) (SQL Server)– SQL Server Version

Score >= 1

OK. The SQL Server version running meets or exceeds the Sitecore recommended requirements.

Score < 1

Error. The SQL Server version does not meet the Sitecore recommended requirements. Sitecore highly recommends running SQL Server 2008 R2 x64.

Sitecore Analytics (OMS 1.x) (SQL Server)– Processor Cores

Score >= 1

OK. The number of CPU cores installed on the server meets or exceeds the Sitecore recommended requirements of 8 processor cores.

Score < 1

OK. The number of CPU cores installed on the server does not meet the Sitecore recommended requirements of 8 processor cores.

Sitecore Analytics (OMS 1.x) (SQL Server)– RAM

Score >= 1

OK. The amount of RAM installed on the server meets or exceeds the Sitecore recommended requirements of 16 GB RAM.

Score < 1

OK. The amount of RAM installed on the server does not meet the Sitecore recommended requirements of 16 GB RAM.

Sitecore Analytics (OMS 1.x) (SQL Server) - Disks

Score >= 1

OK. Data, Logs, and temp space are on separate disks, meeting Sitecore recommended requirements.

Score < 0

Error. Data, Logs, and temp space are not on separate disks, not meeting Sitecore recommended requirements.

Chapter 4

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

4.1 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.

4.1.1 Required Skills

- A working knowledge of SQL Server Management Studio.

4.1.2 Symptoms

- Inability to run scripts to improve performance.

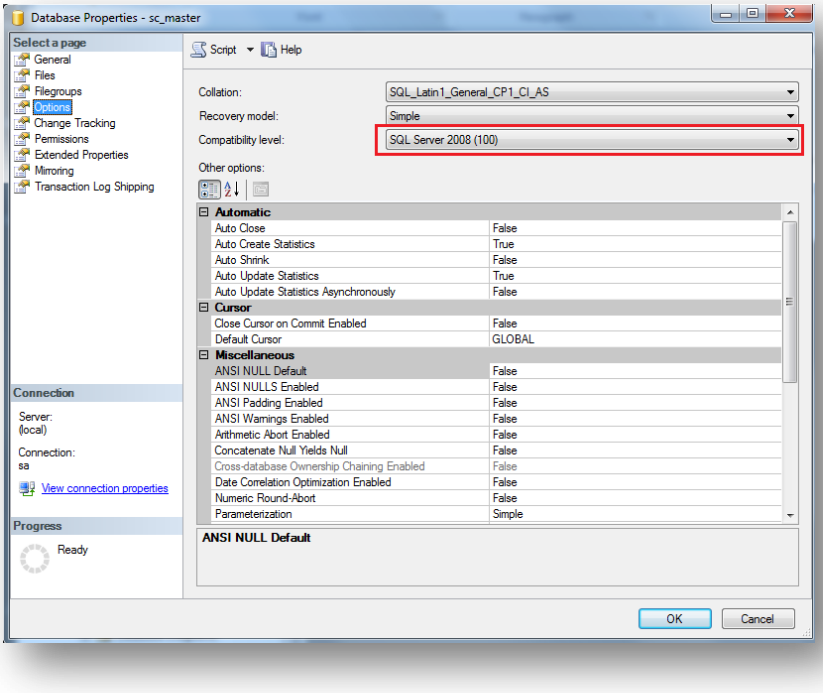
4.1.3 Procedure to Check the Database Compatibility Level

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.

4.1.4 Understanding the Results

The output looks like this:



4.1.5 Sitecore Recommendation

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

4.1.6 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.

Report Findings

Record the Results

Compatibility Level property is set to SQL Server 2008(100): __Yes __No

Compatibility Level property is set to **SQL Server 2008(100)**:

OK. The Compatibility Level property is set to the correct value to take advantage of new T-SQL features.

Compatibility Level property is not set to **SQL Server 2008(100)**:

Error. The Compatibility Level property is not set to SQL Server 2008(100). Sitecore recommends that the Compatibility Level should be set to SQL Server 2008(100) to be able to take advantage of new T-SQL features.

4.2 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.

4.2.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

4.2.2 Symptoms

- Longer times required connecting to the database.

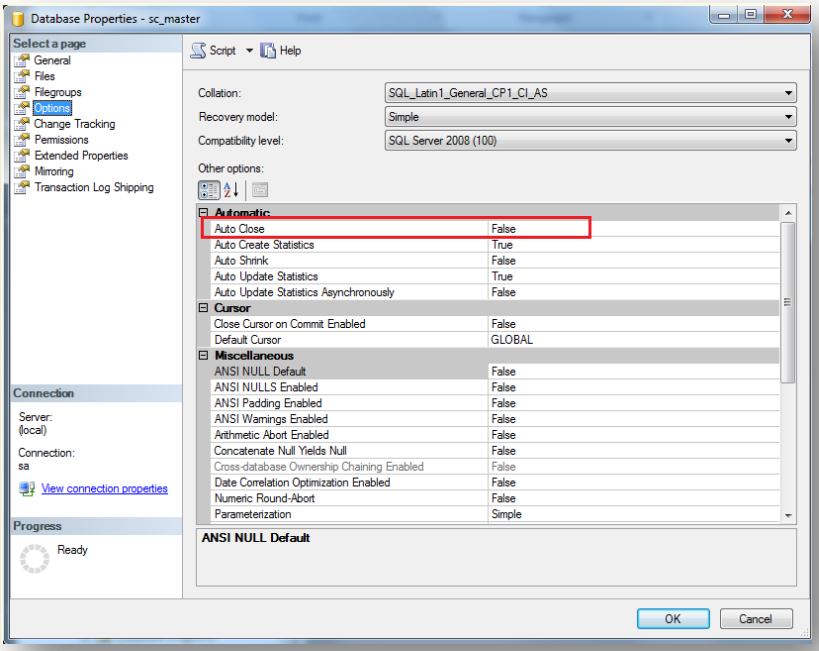
4.2.3 Procedure to check for Auto Close Property value

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.

4.2.4 Understanding the Results

The output looks like this:



4.2.5 Sitecore Recommendation

We recommend that the **Auto Close** property be set to **False**.

4.2.6 How to Solve

In order 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.

Report Findings

Record the Results

Auto Close property is set to: __True __False

Auto Close property is set to **False**:

OK. The Auto Close property is set to the correct value for best performance.

Auto Close property is set to **True**:

Error. The Auto Close property is set to the incorrect value for best performance. Sitecore recommends that the Auto Close property be set to 'False'.

4.3 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.

4.3.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

4.3.2 Symptoms

- Performance degradation.

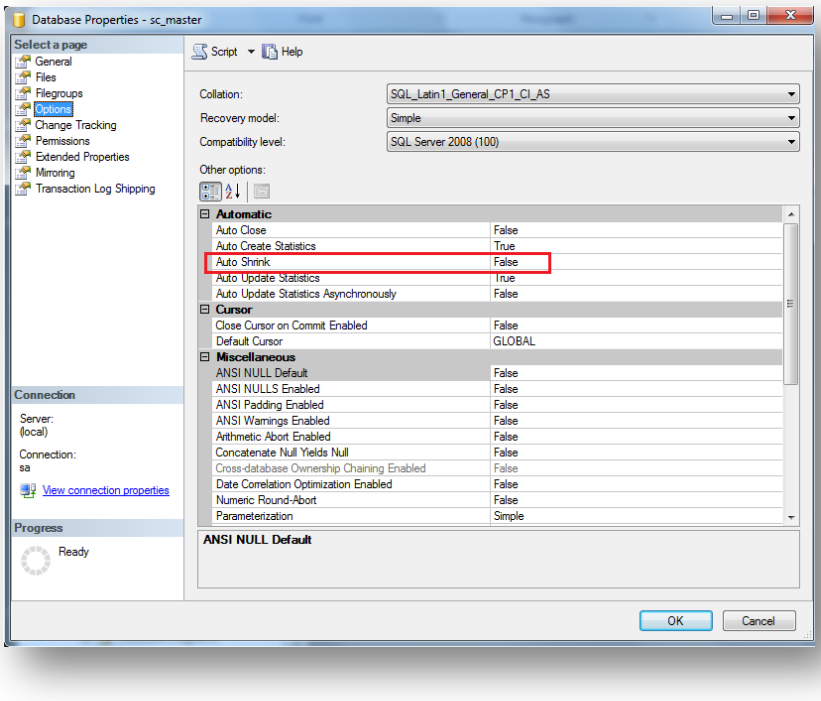
4.3.3 Procedure to Check the Auto Shrink Property Value

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.

4.3.4 Understanding the Results

The output looks like this:



4.3.5 Sitecore Recommendation

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

4.3.6 How to Solve

In order 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.

Report Findings

Record the Results

Auto Shrink property is set to: __True __False

Auto Shrink property is set to **False**:

OK. The Auto Shrink property is set to the correct value for best performance.

Auto Shrink property is set to **True**:

Error. The Auto Shrink property is set to the incorrect value for best performance. Sitecore recommends that the Auto Shrink property be set to 'False'.

4.4 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.

4.4.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

4.4.2 Symptoms

- Performance degradation during recovery intervals.

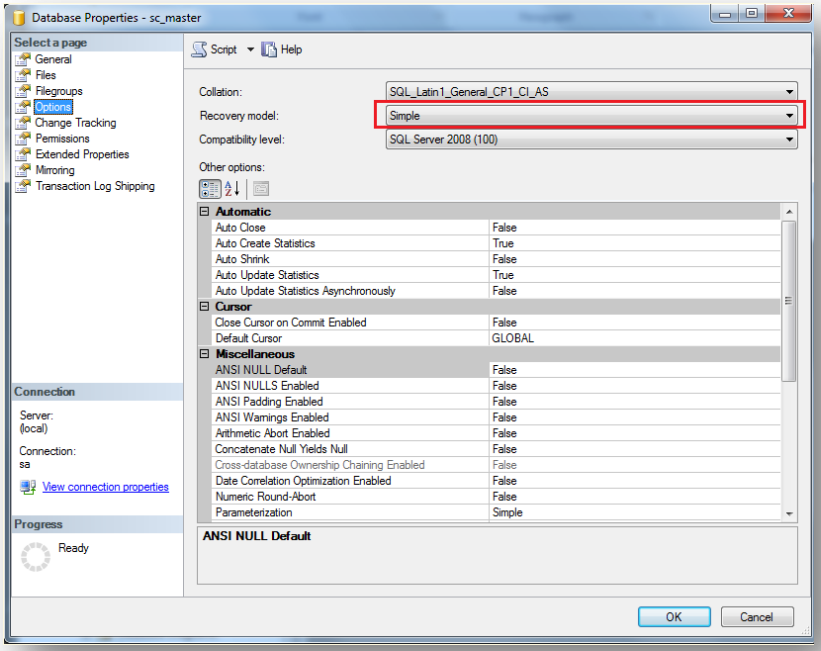
4.4.3 Procedure to Check the Recovery Model Property Value

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.

4.4.4 Understanding theResults

The output looks like this:



4.4.5 Sitecore Recommendation

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

4.4.6 How to Solve

In order 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.

Report Findings

Record the Results

Recovery Model property is set to **Simple**: __Yes __No

Recovery Model property is set to **Simple:**

OK. The Recovery Model property is set to the correct value for optimal database performance.

Recovery Model property is not set to **Simple:**

Error. The Recovery Model property is not set to Simple. Sitecore recommends that the Recovery Model should be set to Simple for optimal database performance.

Chapter 5

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.

This chapter contains the following sections:

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

5.1 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.

5.1.1 Required Skills

- A working knowledge the Sitecore configuration files.

5.1.2 Symptoms

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

5.1.3 Initial Values

The table shown below 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)		
Master	Prefetch	200 MB
Master	Data	200 MB

Environment / Target	Cache	Value
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

5.1.4 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.

5.1.5 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.
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.

5.1.6 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.

5.1.7 Sitecore Recommendation

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

5.1.8 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.

Report Findings

Record the Results

The prefetch, data, item, and html output caches have been set to initial values (based on environment) ___YES ____NO

The prefetch, data, item, and html output caches have been set to initial values = YES

Ok. The results show that the database caches have been set prior to tuning.

The prefetch, data, item, and html output caches have been set to initial values = NO

Error. The results show that the database caches have not been set prior to tuning. Sitecore recommends that the prefetch, data, item, and html output caches be set to initial values prior to performing cache tuning.

5.2 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.

5.2.1 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.

5.2.2 Symptoms

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

5.2.3 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.

5.2.4 Prerequisites

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

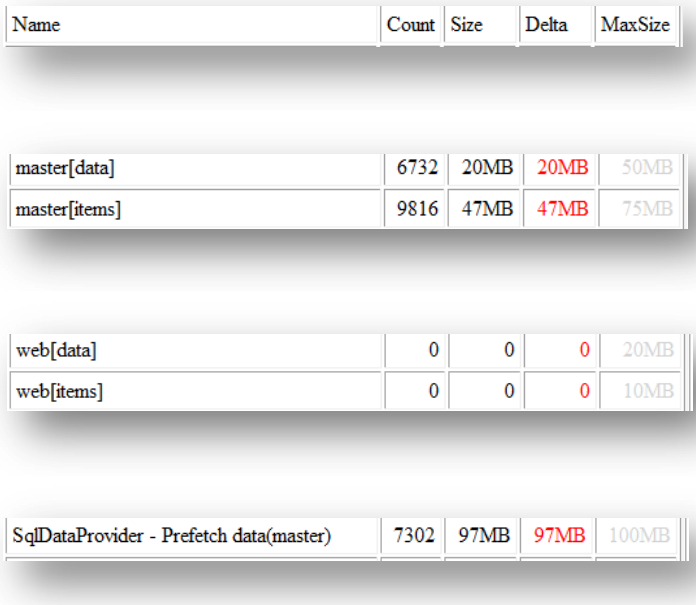
A load generation tool and script is needed to exercise the caches. For more information about programs to generate web load, see the *Using WCAT to Generate Load* or *Using Web Performance Load Tester*.

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

5.2.5 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], 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.

5.2.6 Understanding the Results

[Include Screen shots of cache.aspx page]

5.2.7 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.

5.2.8 How to Solve

Follow the Procedure for tuning the Sitecore database caches described above.

Report Findings

Record the Results

The Sitecore database caches required tuning ___YES ____NO

The Sitecore database caches required tuning = NO

Ok. The results show that the database caches have been properly tuned and meet Sitecore recommendations.

The Sitecore database caches required tuning = YES

Error. The results show that the database caches have not been tuned. Sitecore recommends that the Sitecore database cache size(s) be between 70% and 80% of their MaxSize, without seeing constant fluctuations to the Delta value.

5.3 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

5.3.1 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.

5.3.2 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>`.
- `<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>`.

5.3.3 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.

5.4 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

5.4.1 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.

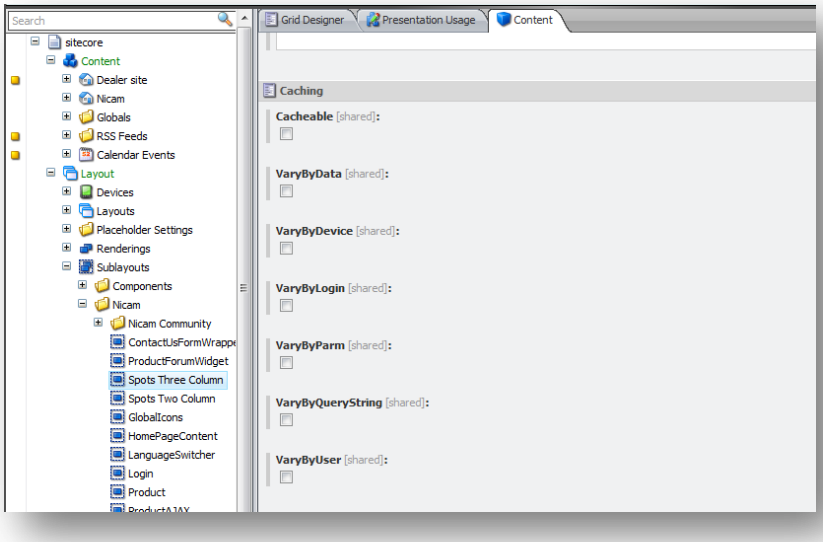
The default behavior of the layout engine is to not cache presentation components. To take advantage of the Sitecore output caching, presentation components must be configured properly. This section provides guidance and recommended practices on how to do so.

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.

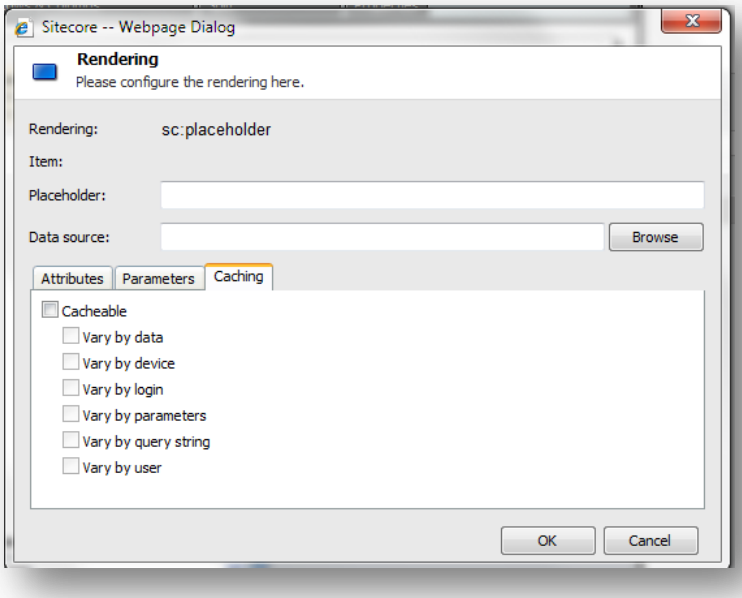
5.4.2 Where to Configure Cache Settings and Which Cache Settings Apply

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

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



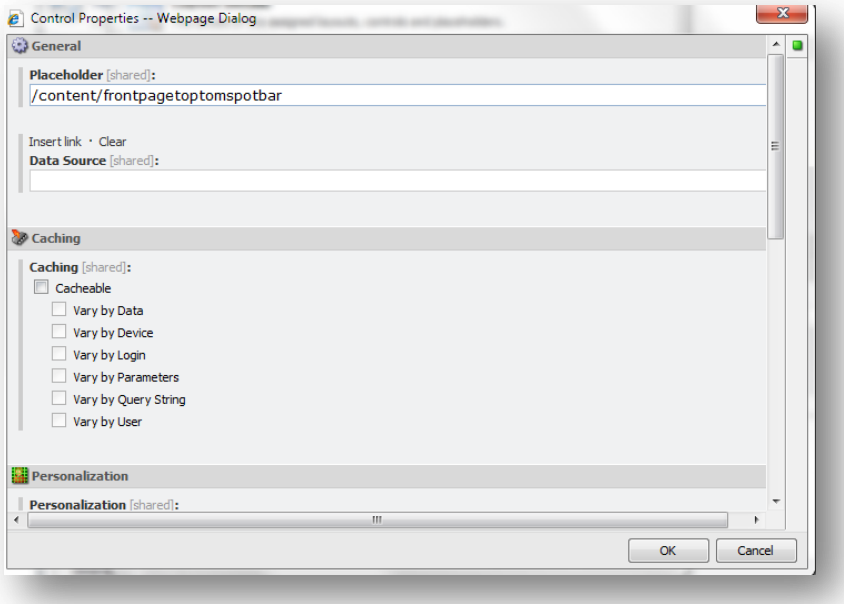
- In the properties of the presentation component when you statically bind it to a layout or sublayout.



- 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>
```

- 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.

5.4.3 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.
  - **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>`

5.4.4 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

Chapter 6

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)

6.1 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.

6.1.1 Required Skills

- A working IIS Manager.

6.1.2 Symptoms

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

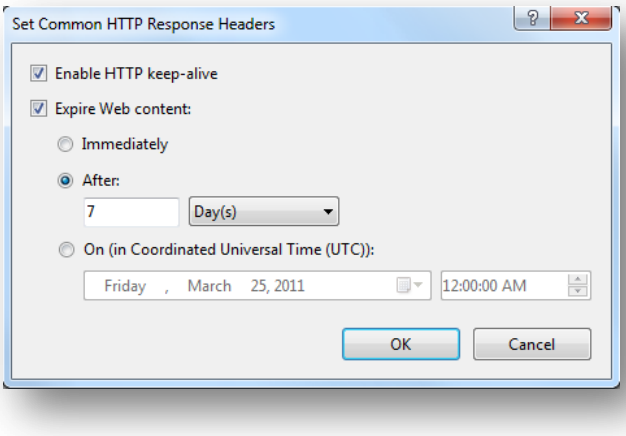
6.1.3 Procedure

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**...

6.1.4 Understanding the Results

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



6.1.5 Sitecore Recommendation

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

6.1.6 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**.

Report Findings

Record the Results

HTTP keep-alive is enabled ___YES ____NO

HTTP keep-alive is enabled = YES

Ok. The results show that the HTTP keep-alive is enabled, following Sitecore recommended practices.

HTTP keep-alive is enabled = NO

Error. The results show that the HTTP keep-alive is not enabled. Sitecore recommends that the HTTP keep-alive be enabled, reducing the number of connections required to be opened.

6.2 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 procedures shown below are for IIS7.x. For information about how to enable the Expire Web content header in previous versions, refer to Microsoft's documentation.

6.2.1 Required Skills

- A working IIS Manager.

6.2.2 Symptoms

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

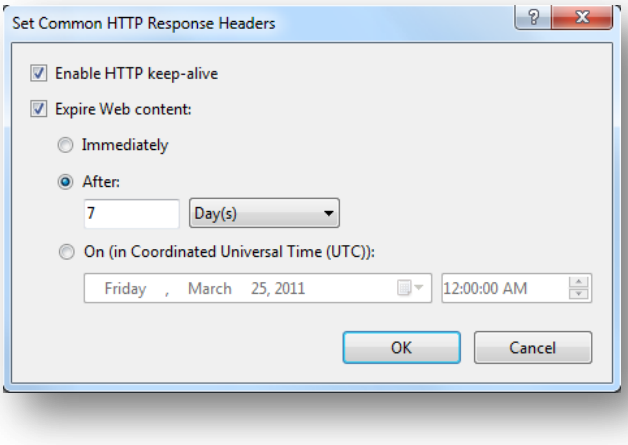
6.2.3 Procedure

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**...

6.2.4 Understanding the Results

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



6.2.5 Sitecore Recommendation

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

6.2.6 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**.

Report Findings

Record the Results

Expire Web content header is enabled and set to After ___YES ____NO

Expire Web content header is enabled and set to After = YES

Ok. The results show that the Expire Web content header is enabled and set to After, following Sitecore recommended practices.

Expire Web content header is enabled and set to After = NO

Error. The results show that the Expire Web content header is not enabled or set to After. Sitecore recommends that the Expire Web content be enabled, Set to "After" 30 days.

6.3 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.

6.3.1 Required Skills

- A working IIS Manager.

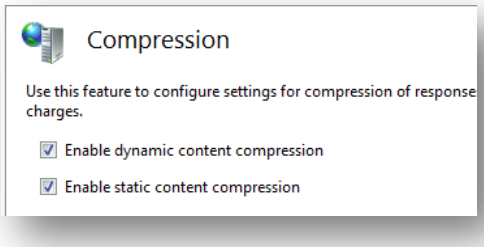
6.3.2 Symptoms

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

6.3.3 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 example below shows that the static content compression is enabled.



6.3.4 Sitecore Recommendation

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

6.3.5 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).
4. Select the **Enable static content compression** check box.
5. Click **Apply**.

Report Findings

Record the Results

IIS Static content compression is enabled ___YES ____NO

IIS Static content compression is enabled = YES

Ok. The results show that the IIS static content compression is enabled, following Sitecore recommended practices.

IIS Static content compression is enabled = NO

Error. The results show that IIS static content compression is not enabled. Sitecore recommends that the IIS static content compression be enabled.

6.4 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

6.4.1 Required Skills

- A working IIS Manager.

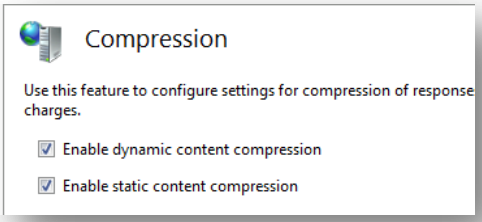
6.4.2 Symptoms

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

6.4.3 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 example below shows that the dynamic content compression is enabled.



6.4.4 Sitecore Recommendation

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

6.4.5 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 **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**.

Report Findings

Record the Results

IIS Dynamic content compression is enabled ___YES ____NO

IIS Dynamic content compression is enabled = YES

Ok. The results show that the IIS dynamic content compression is enabled, following Sitecore recommended practices.

IIS Dynamic content compression is enabled = NO

Error. The results show that IIS dynamic content compression is not enabled. Sitecore recommends that the IIS dynamic content compression be enabled on systems where CPU utilization is not at a constant high level.

Chapter 7

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

7.1 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.

7.1.1 Required Skills

- A working knowledge of reading the Sitecore logs.

7.1.2 Symptoms

- Slow performance while using the Sitecore Content Editor.

7.1.3 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 Log Parser Lizard GUI.

To parse the log file using Log Parser Lizard GUI:

1. Launch **Log Parser Lizard GUI**.
2. Create a **New Query**.
3. Change the Input Log Format to **Multiline Regex (Log4Net)** Input Format.

   Run the following script (change the #logs location#, for example “c:\logs*.txt”):

```
SELECT *
FROM #Logs Location#
        WHERE Message LIKE '%Long running operation: Running Validation Rules%'
```

7.1.4 Understanding the Results

The output looks like this:

Filename	RowNumber	ProcessID	Time	LogType	Message
C:\scinternet\sc\log.20110209.100803.txt	7,668	2976	13:56:19	WARN	Long running operation: Running Validation Rules
C:\scinternet\sc\log.20110209.100803.txt	10,576	2588	14:56:08	WARN	Long running operation: Running Validation Rules
C:\scinternet\sc\log.20110209.100803.txt	13,685	4020	15:36:42	WARN	Long running operation: Running Validation Rules
C:\scinternet\sc\log.20110210.150537.txt	330	1416	15:06:24	WARN	Long running operation: Running Validation Rules
C:\scinternet\sc\log.20110210.150537.txt	349	1416	15:06:27	WARN	Long running operation: Running Validation Rules

The above table shows that 5 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.

7.1.5 Sitecore Recommendation

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

7.1.6 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).

Report Findings

Record the Results

The log files show long running validation rules __Yes __No

The log files show long running validation rules = No

OK. No long running validation rules are present in your Sitecore solution.

The log files show long running validation rules = Yes:

Error. The log files show that long running validation rules are present. Sitecore recommends investigating and disabling those validation rules producing warnings.

7.2 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.

7.2.1 Required Skills

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

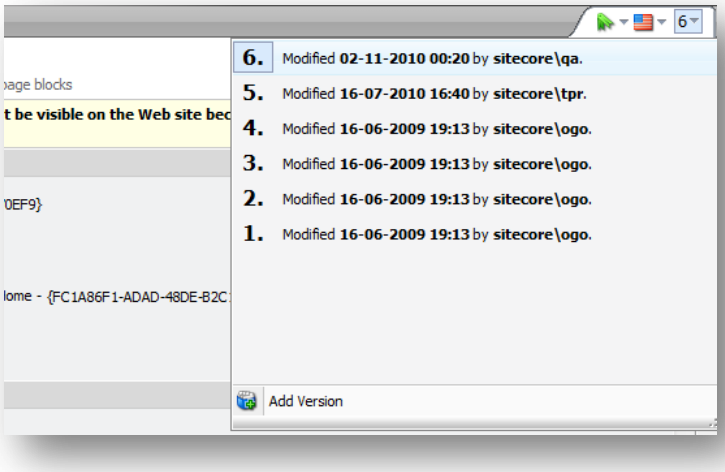
7.2.2 Symptoms

- Slow performance while using the Sitecore Content Editor.

7.2.3 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.

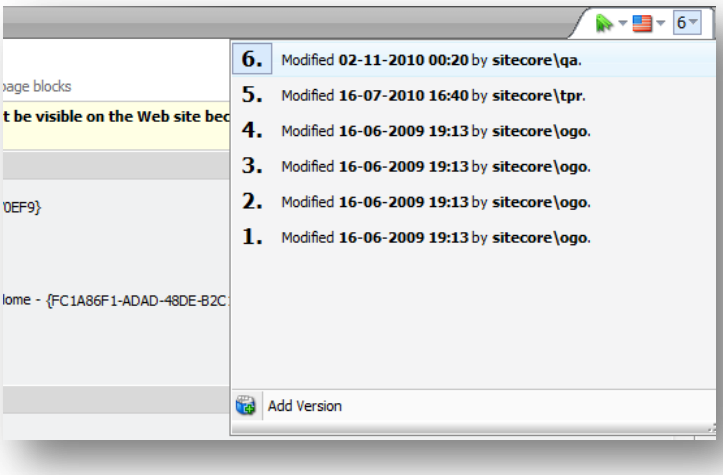
1. Launch the **Sitecore Content Editor**.
2. In the content tree, navigate to the items that are in question.
3. In the upper right hand corner of the item, there is a drop down showing the number of versions for the item being viewed:



4. Note the number of versions for the item (anything over 10 should be flagged as excessive).
5. Repeat the process for other items in question.

7.2.4 Understanding the Results

The output looks like this:



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

7.2.5 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.

7.2.6 How to Solve

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

Report Findings

Record the Results

Greater than 10 versions exists for one or more items __Yes __No

Greater than 10 versions exists for one or more items = No

OK. No items have greater than 10 versions.

Greater than 10 versions exists for one or more items = Yes:

Error. One or more items have been found to have greater than 10 versions. Sitecore recommends keeping the number of versions for any item to 10 or less, as well running the Version Manager shared source module to maintain the number of stored versions.

7.3 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.

7.3.1 Required Skills

- A working knowledge the Sitecore content editor.

7.3.2 Symptoms

- Slow performance while using the Sitecore Content Editor.

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

```
                {
                  Response.Write("<br />");
                }
              }
            }
          }
        }
```

7.3.4 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.

7.3.5 Sitecore Recommendation

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

Report Findings

Record the Results

Greater than 100 items exists under any parent node __Yes __No

Greater than 100 items exists under any parent node = No

OK. No parent nodes have greater than 100 items.

Greater than 100 items exists under any parent node = Yes:

Error. One or more parent nodes have greater than 100 items. Sitecore recommends that 100 or fewer items under any parent node.

7.4 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.

7.4.1 Required Skills

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

7.4.2 Symptoms

- Slow performance while using the Sitecore Content Editor.

7.4.3 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 due to not having to render as much information per item.

7.4.4 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.

7.4.5 ContentEditor.CheckHasChildrenOnTreeNodes setting

This setting determines if the editor uses the HasChildren method when rendering a tree node. Setting this value to false may increase performance. Additional testing should be performed to see in any benefit has been realized.