Sitecore CMS 7.0 or later
CMS Diagnostics Guide

A developer's guide to diagnosis of Sitecore CMS performance
# Table of Contents

Chapter 1  Introduction .............................................................................................................. 4
Chapter 2  Diagnostic Procedures .......................................................................................... 5
  2.1  Browser Page Test ........................................................................................................ 6
    2.1.1  Setup .................................................................................................................... 6
    2.1.2  How to Use the AOL Pagetext Browser Plug-in .................................................. 6
      Capturing Information .................................................................................................. 7
      First Pass — Cleared Browser Cache ........................................................................ 7
      Second Pass — Refresh Browser Page ......................................................................... 10
      Third Pass — Recently Visited Webpages .................................................................. 10
    2.1.3  Interpreting the Results ....................................................................................... 11
      Waterfall Tab ............................................................................................................. 11
      Checklist Tab ............................................................................................................ 12
      Information about Solving Failures and Warnings ..................................................... 14
      Optimization Report Tab ............................................................................................ 14
      Load Details Tab ........................................................................................................ 14
  2.2  Investigating Page Performance — Using the IIS Logs ............................................. 15
    2.2.1  Required Skills ..................................................................................................... 15
    2.2.2  Symptoms ............................................................................................................ 15
    2.2.3  Talk to the Partner / Customer .......................................................................... 15
    2.2.4  Procedure to Query IIS Logs for Long Running Requests .............................. 15
    2.2.5  Understanding the Results ................................................................................. 16
    2.2.6  Sitecore Recommendation .................................................................................. 17
      Report Findings ........................................................................................................ 18
  2.3  Rendering Performance ............................................................................................... 19
    2.3.1  Required Skills ..................................................................................................... 19
    2.3.2  Symptoms ............................................................................................................ 19
    2.3.3  Procedure to Use the Sitecore Stats Page .......................................................... 19
    2.3.4  Understanding the Results ................................................................................. 20
    2.3.5  Sitecore Recommendation .................................................................................. 20
      Report Findings ........................................................................................................ 21
  2.4  Investigating Sitecore Memory Usage ....................................................................... 22
    2.4.1  Required Skills ..................................................................................................... 22
    2.4.2  Symptoms ............................................................................................................ 22
    2.4.3  Parsing the Sitecore Log(s) using Sitecore Log Analyzer ................................ 22
      Getting the Values of Process/Private Bytes and # Bytes in all Heaps Counters .... 23
      Graphing the Results ................................................................................................. 24
    2.4.4  Understanding the Results (Graph) .................................................................. 27
    2.4.5  Notes .................................................................................................................... 30
  2.5  Sitecore Pipeline Profiling ........................................................................................... 32
    2.5.1  Required Skills ..................................................................................................... 32
    2.5.2  Procedure to Use the Sitecore Pipeline Profiling Page ................................. 32
    2.5.3  Overview ............................................................................................................. 32
    2.5.4  Usage ................................................................................................................... 33
    2.5.5  Understanding the Results ................................................................................. 33
  2.6  Sitecore Debugger ........................................................................................................ 35
    2.6.1  Procedure to activate Sitecore Debugger ........................................................ 35
    2.6.2  Usage ................................................................................................................... 36
  Chapter 3  Search Diagnostics .............................................................................................. 40
  3.1  FillDB .......................................................................................................................... 41
    3.1.1  Generating Items Using the FillDB Page .......................................................... 41
  3.2  Verbose Logging .......................................................................................................... 43
  3.3  LinqScratchPad ............................................................................................................ 44
    3.3.1  Required Skills ..................................................................................................... 44
    3.3.2  Usage .................................................................................................................... 44

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2013 Sitecore. All rights reserved.
Chapter 1

Introduction

This guide is designed as a companion guide to the Sitecore CMS Performance Guide. It provides you with a series of diagnostic procedures that can help you identify performance issues, as well as a means to measure performance gains through tuning.

This manual contains the following chapters:

- Chapter 1 — Introduction
- Chapter 2 — Diagnostic Procedures
- Chapter 3 — Search Diagnostics
Chapter 2

Diagnostic Procedures

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

The procedures defined are complementary 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 Sitecore Memory Usage
- Sitecore Pipeline Profiling
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 it in its default location. The AOL Pagetest plug-in appears in the Tools menu and works with IE 7 and later.

Note
If IE was opened during the installation, 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.
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 is visited from the browser being used.
  
  To clear the browser cache for IE 8 and later:
  1) In the Safety menu, click Delete browsing history...
  2) Clear Preserve Favorites website data, and select Temporary Internet Files, Cookies, and History.

  To clear the browser cache for IE 7:
  1) In the Tools menu, click Internet Options.
  2) Under Browsing History, click Delete...
  3) To delete your cache, click Delete files...
  4) 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 code appears, a request is still made for an object, even though no download occurs.

- Accessing a recently visited webpage — typing 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 Cache

1. Launch IE.
2. Clear the browser cache.
3. Launch AOL Pagetest.
4. Type in the URL you wish to test — for example: http://www.sitecore.net.
5. The **AOL Pagetest** window contains 4 tabs — **Waterfall, Checklist, Optimization Report**, and **Load Details**.

![Diagram](image)

Observations:

- 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 timeline is pushed to the left, due to a single request being made, shortening the time it takes 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 them 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 — Recently Visited Webpages.

Third Pass — Recently Visited Webpages

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 them to the results from the first and second passes.

![Waterfall tab graphic]

**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 — why is it not coming from cache?

  Upon further investigation it was found that this request is a rotating image, which changes across visits. It is OK not to be in cache.

### 2.1.3 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 across the top.

The colors that make up a request / response are:

- **DNS Lookup** — The time it takes to look up the IP Address for the requested URL.
- **Initial Connection** — The time that it takes to go from the client to the server to open up a socket.

  If an initial connection is made on every request, this could indicate that HTTP keep-alive has not been enabled on the server.
- **Time to First Byte** — The time that it takes from making the request to receiving the first byte of the object. If the *Time to First Byte* times are extended, this could indicate that there is a problem at the server for example, performance issues, or network issues.

- **Content Download** — The time required to download an object. If the Content Download times are extended, this could indicate the presence 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** — The green vertical line shown on the Waterfall tab. 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** — 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 website 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 describes each column in the **Checklist** tab, what objects they apply to, as well as what is checked:

<table>
<thead>
<tr>
<th>Column</th>
<th>Objects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cache Static</strong></td>
<td>Applicable Objects</td>
<td>Any non-html object with a mime type of &quot;text/<em>&quot;, &quot;javascript/</em>&quot; or &quot;image/*&quot; that does not explicitly have an Expires header of 0 or -1, a cache-control header of &quot;private&quot;, &quot;no-store&quot; or &quot;no-cache&quot; or a pragma header of &quot;no-cache&quot;.</td>
</tr>
<tr>
<td>What is checked</td>
<td></td>
<td>An &quot;Expires&quot; header is present (and is not 0 or -1) or a &quot;cache-control: max-age&quot; 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).</td>
</tr>
<tr>
<td><strong>Use A CDN</strong></td>
<td>Applicable Objects</td>
<td>All static non-html content (css, js and images)</td>
</tr>
<tr>
<td>What is checked</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Combine CSS/JS</strong></td>
<td>Applicable Objects</td>
<td>All css and javascript objects.</td>
</tr>
<tr>
<td>What is checked</td>
<td></td>
<td>If multiple files of the same type are served, each additional css file beyond 1 subtracts 5 percent and each Javascript file beyond the first subtracts 10 percent.</td>
</tr>
<tr>
<td><strong>GZIP Text</strong></td>
<td>Applicable Objects</td>
<td>All objects with a mime type of &quot;text/<em>&quot; or &quot;javascript/</em>&quot;</td>
</tr>
<tr>
<td>What is checked</td>
<td></td>
<td>Transfer-encoding is checked to see if it is gzip. If it is not, 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).</td>
</tr>
<tr>
<td><strong>Compress Images</strong></td>
<td>Applicable Objects</td>
<td>Any object with a mime type of &quot;image/*&quot;.</td>
</tr>
</tbody>
</table>
| 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.                                                                                                       |
### Information about 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.
- 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 exist 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 website. This information is used to determine if the website has the capacity to handle load during peak usage.

2.2.4 Procedure to Query IIS Logs for Long Running Requests

Information about installing the Log Parser can be found at http://www.microsoft.com/en-us/download/details.aspx?id=24659

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 the command line in a directory where the Log Parser is installed.
2. Run the following command — change the #logs location# and #output file#: 

```
logparser -i:IISW3C -o:CSV
"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]
INTO #output file#
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

Open the output file in Excel and look at the results in two parts. The first is a series of long running requests that are all coming from the same client IP address. For this purpose you can use sorting or filtering functionality in Excel. This could be an indication of in-house testing. Or could show the IP addresses with the slow or limited bandwidth. The second is a more typical set of results coming from external traffic to the website.

Results Part 1

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 to track down possible network issues based on location.
Results Part 2

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

If there is a number of URLs that are above the threshold, in our case 4 seconds, that occur during peak operation this could indicate a website 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 website are: _______________________________________

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

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>URI</th>
<th>Client IP</th>
<th>Status Code</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There are requests that 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 URLs 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

You can use the Sitecore stats page to collect information about rendering.

The stats page provides the following information about the various renderings used in a page, or throughout the website — depending when the stats page is observed.

- Rendering — The name of the rendering.
- Site — The name of the website 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 you open the stats page in a separate tab or browser window, so that you can keep it open while you use another tab or window to navigate through the website 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**
To collect information about caching and averages, make several calls to the page.

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 the information, right click the table and then click Export to Microsoft Excel.

### 2.3.4 Understanding the Results

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rendering</td>
<td>Site</td>
<td>Count</td>
<td>From cache</td>
<td>Avg. time (ms)</td>
<td>Max. time</td>
<td>Max. time</td>
</tr>
<tr>
<td>Placeholder: content</td>
<td>nicam</td>
<td>20</td>
<td>0</td>
<td>185.3346</td>
<td>1670.1252</td>
<td></td>
</tr>
<tr>
<td>Sublayout: /layouts/Nicam/HomePageContent.ascx</td>
<td>nicam</td>
<td>2</td>
<td>0</td>
<td>1080.6818</td>
<td>1669.1879</td>
<td></td>
</tr>
<tr>
<td>Placeholder: rightcolumn</td>
<td>nicam</td>
<td>15</td>
<td>0</td>
<td>142.9569</td>
<td>1417.7889</td>
<td></td>
</tr>
<tr>
<td>JSL/Nicam/NewSpots.xsl</td>
<td>nicam</td>
<td>2</td>
<td>0</td>
<td>938.395</td>
<td>1415.1291</td>
<td></td>
</tr>
<tr>
<td>Sublayout: /layouts/Nicam/ThreeColumnContent.ascx</td>
<td>nicam</td>
<td>13</td>
<td>0</td>
<td>86.9666</td>
<td>362.6316</td>
<td></td>
</tr>
<tr>
<td>Placeholder: centercolumn</td>
<td>nicam</td>
<td>18</td>
<td>0</td>
<td>50.0488</td>
<td>228.2898</td>
<td></td>
</tr>
<tr>
<td>Sublayout: /layouts/Nicam/Product Forums.ascx</td>
<td>nicam</td>
<td>1</td>
<td>0</td>
<td>327.9865</td>
<td>327.9865</td>
<td></td>
</tr>
<tr>
<td>Sublayout: /layouts/Nicam/TwoColumnContent.ascx</td>
<td>nicam</td>
<td>5</td>
<td>0</td>
<td>82.6878</td>
<td>274.0745</td>
<td></td>
</tr>
<tr>
<td>Sublayout: /layouts/Nicam/Contacts/FormWrapper.ascx</td>
<td>nicam</td>
<td>1</td>
<td>0</td>
<td>290.0593</td>
<td>260.0593</td>
<td></td>
</tr>
<tr>
<td>FormRenderer (FormRender)</td>
<td>nicam</td>
<td>1</td>
<td>0</td>
<td>290.0593</td>
<td>260.0593</td>
<td></td>
</tr>
<tr>
<td>Placeholder: forum-content</td>
<td>nicam</td>
<td>1</td>
<td>0</td>
<td>178.4445</td>
<td>178.4445</td>
<td></td>
</tr>
<tr>
<td>Sublayout: /sitescore modules/Web/YAH/YAF_Forum.ascx</td>
<td>nicam</td>
<td>1</td>
<td>0</td>
<td>178.4049</td>
<td>178.4049</td>
<td></td>
</tr>
<tr>
<td>Placeholder: phxsl</td>
<td>nicam</td>
<td>2</td>
<td>0</td>
<td>86.4851</td>
<td>182.7259</td>
<td></td>
</tr>
<tr>
<td>JSL/Themehandler/Flash_XMLOutput.xsl</td>
<td>nicam</td>
<td>2</td>
<td>0</td>
<td>86.4851</td>
<td>182.7259</td>
<td></td>
</tr>
<tr>
<td>JSL/Nicam/logo.xsl</td>
<td>nicam</td>
<td>21</td>
<td>7</td>
<td>12.5509</td>
<td>146.5511</td>
<td></td>
</tr>
<tr>
<td>Placeholder: frontpagebottomspotbar</td>
<td>nicam</td>
<td>2</td>
<td>0</td>
<td>84.1093</td>
<td>142.2216</td>
<td></td>
</tr>
<tr>
<td>Sublayout: /layouts/Nicam/Spots/Three Column.ascx</td>
<td>nicam</td>
<td>4</td>
<td>0</td>
<td>42.0559</td>
<td>142.2195</td>
<td></td>
</tr>
<tr>
<td>Placeholder: spotbarcenter</td>
<td>nicam</td>
<td>4</td>
<td>0</td>
<td>33.8659</td>
<td>121.6914</td>
<td></td>
</tr>
<tr>
<td>JSL/Nicam/ResponsiveFooter.ascx</td>
<td>nicam</td>
<td>6</td>
<td>0</td>
<td>32.8522</td>
<td>121.6471</td>
<td></td>
</tr>
<tr>
<td>JSL/Nicam/ResponsiveContact.ascx</td>
<td>nicam</td>
<td>9</td>
<td>0</td>
<td>13.5972</td>
<td>111.2867</td>
<td></td>
</tr>
<tr>
<td>JSL/Nicam/ResponsiveLogo.ascx</td>
<td>nicam</td>
<td>21</td>
<td>0</td>
<td>17.2713</td>
<td>106.6543</td>
<td></td>
</tr>
<tr>
<td>JSL/Nicam/ResponsiveLogo.ascx</td>
<td>nicam</td>
<td>12</td>
<td>0</td>
<td>17.6432</td>
<td>100.6103</td>
<td></td>
</tr>
<tr>
<td>Placeholder: frontflash</td>
<td>nicam</td>
<td>2</td>
<td>0</td>
<td>45.6583</td>
<td>89.6989</td>
<td></td>
</tr>
</tbody>
</table>

The following observations can be made from this table:

- All renderings that have a 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 you enable and configure Rendering (HTML Output) caching.
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 website 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 Sitecore Memory Usage

The Sitecore system contains a series of performance counters that are logged to the Sitecore log files 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, Sitecore Log Analyzer 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 Sitecore Log Analyzer.
- 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 Sitecore Log Analyzer

For more information about installing and setting up the Sitecore Log Analyzer, see http://sdn.sitecore.net/Resources/Tools/Log Analyzer.aspx
This task requires retrieving the values of the necessary health monitor counters from the log files and exporting them in CSV file.

The exported results are opened in Excel so that they can be graphed and compared.

**Getting the Values of Process\Private Bytes and # Bytes in all Heaps Counters**

The Health Monitor tab of Sitecore Log Analyzer allows you to see and export all health monitor counters of the parsed log files in a convenient format.

1. Launch Sitecore Log Analyzer.
2. Select logs for analyzing.
3. Click Analyze / Refresh.
4. Go to the Health Monitor tab and select the necessary counters — Process\Private Bytes and .NET CLR Memory\# Bytes in all Heaps.
5. Right click on the table and then click **Export Table To, CSV File**.
6. Give the csv file a meaningful name and open it in **Excel**.

---

**Graphing the Results**

There are several ways to create graphs in **Excel**. The following procedure describes how to create a line graph and compare the Process\Private Bytes results set to the .net CLR Memory\# Bytes in all Heaps.

1. In **Excel**, on the **Insert** tab, in the **Charts** group, click **Line** and then click **2D Line**.

2. In the **Data** group, click **Select Data**.
   
   If something was selected on the sheet, Excel attempts to create graph of it.
3. To clear the graph, remove any series and clear the Chart data range. The Select Data Source dialog box appears.

![Select Data Source dialog box]

4. Click Add.

5. For the first series, name the series Private Bytes and select the data range from the Process\Private Bytes column — column B.

![Add Series dialog box]

6. Set the Horizontal — x-axis — labels to the Timestamp column — column A — and click Edit and select column A from sheet 1:

![Axis Labels dialog box]
7. Click **OK**.

8. Click **Add** and do the same for the second series — Bytes in all Heaps. Select the data range from the `.net CLR Memory\# Bytes in all Heaps` column — column C.
   
   You do not need to set the Horizontal — x-axis — labels again.

9. Click **OK**.
2.4.4 Understanding the Results (Graph)

Graph 1

- This graph illustrates an example of a potential memory leak in the managed memory space. The five big dips could represent the clearing of large Sitecore caches or 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 Bytes in all Heaps are tending to increase in parallel. This could be a symptom of a memory leak, but it could also be normal reaction to increased load on the server requiring increased memory consumption and potential cache clearing. Further investigation would be required.

- 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 raise to a stable point and remain there.

Note 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 are remaining constant, this would indicate that there is a potential leak in the unmanaged memory space and further investigation would be required.
• Peaks indicate memory allocation. When you move the mouse over a peak, information about the amount of memory allocated is displayed, 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 — in IIS Manager, select the appropriate App Pool, and select Advanced Settings.

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.

In the following image, 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.

2.4.5 Notes

For more information about .net memory usage and how to investigate it see:

2.5 Sitecore Pipeline Profiling

By using the Sitecore pipeline profiling page, metrics related to the performance of core Sitecore pipelines can be monitored and collected. Only pipelines invoked by the Sitecore.Pipelines.CorePipeline.Run() method are subject to profiling when pipeline profiling is enabled.

2.5.1 Required Skills

- A working knowledge of Sitecore pipelines and how they are used.

2.5.2 Procedure to Use the Sitecore Pipeline Profiling Page

This procedure requires that you have permissions to browse to aspx pages in the http://<site>/sitecore/admin directory.

To enable pipeline profiling, set the Pipelines.Profiling.Enabled setting to true in the web.config file or use the App_Config\Sitecore.PipelineProfiling.config include file:

```
<setting name="Pipelines.Profiling.Enabled" value="true" />
```

Browse to the Sitecore pipeline profiling page:

http://<site>/sitecore/admin/pipelines.aspx

2.5.3 Overview

When pipeline profiling is disabled, the page displays a No data message, and instructions on how to enable profiling.

When pipeline profiling is enabled, the page displays a list of the last used processors grouped by pipelines and usage statistics.
2.5.4 Usage

The pipeline profiling page has two buttons: Refresh and Reset.

To retrieve the latest snapshot of profile counter values and render the page, click Refresh.

To reset the pipeline profiling counters, click Reset. After you click Reset, you must refresh the page to get an updated snapshot.

![Pipeline Profiles](image)

2.5.5 Understanding the Results

**Note**

Certain pipelines are executed every time you click Refresh on the pipeline profiling page and therefore increment/impact the results for those pipelines for example, `httpRequestBegin`, `preprocessRequest`, `renderLayout`, `getTranslation`.

The pipeline profiling page contains the following information about Sitecore pipelines:

- The **# Executions** column provides the number of times the pipeline or processor has been executed over the lifetime of the profiler.

- The **% Wall Time** column is applicable to processors only and provides the ratio of execution time for a processor vs the execution time for the pipeline that invoked the processor. The values in this column are helpful in determining which processors consume the most time during pipeline execution.

- The **Wall Time** column provides the aggregate execution time for each pipeline or processor over the lifetime of the profiler. The values in this column are cumulative over time and are mostly useful for calculating average execution time.

- The **Max Wall Time** column provides the maximum time elapsed during the execution of a pipeline or processor. The values in this column represent the maximum execution time over the lifetime of the profiler and can be used to compare peak execution duration to average execution duration.

- The **% CPU** column is applicable to processors only and provides the ratio of CPU time used by a processor vs the CPU time used by the pipeline that invoked the processor.

  **Note**
  
  The % CPU metric is only measured and displayed when the `Pipelines.Profiling.MeasureCpuTime` setting is set to true in the web.config file (or via config include).

    <setting name="Pipelines.Profiling.MeasureCpuTime" value="true" />

- The **Time / Execution** column provides the average execution time for each pipeline or processor over the lifetime of the profiler. The values in this column are valuable when monitoring pipeline/processor performance over a specific time period (e.g. during load or performance testing).
The top 3 worst performing processors — by execution/wall time — are marked with numbered icons.

<table>
<thead>
<tr>
<th>PreprocessRequest</th>
<th>1117</th>
<th>308.20</th>
<th>16.30</th>
<th>0.27</th>
<th>0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sitecore.Pipelines.PreprocessRequest.SuspendFormValidation.Process</td>
<td>1117</td>
<td>1.49</td>
<td>4.29</td>
<td>0.05</td>
<td>0.01</td>
</tr>
<tr>
<td>Sitecore.Pipelines.PreprocessRequest.NormalizeUriRequest.Process</td>
<td>1117</td>
<td>6.40</td>
<td>19.73</td>
<td>7.91</td>
<td>0.02</td>
</tr>
<tr>
<td>Sitecore.Pipelines.PreprocessRequest.RequestHandler.Process</td>
<td>1117</td>
<td>0.85</td>
<td>2.60</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Sitecore.Pipelines.PreprocessRequest.WebDAVCustomHandler.Process</td>
<td>1117</td>
<td>86.40</td>
<td>266.29</td>
<td>7.99</td>
<td>0.23</td>
</tr>
<tr>
<td>Sitecore.Pipelines.PreprocessRequest.FilterUriExtensions.Process</td>
<td>1117</td>
<td>3.93</td>
<td>12.12</td>
<td>0.31</td>
<td>0.01</td>
</tr>
<tr>
<td>Sitecore.Pipelines.PreprocessRequest.StripLanguage.Process</td>
<td>408</td>
<td>0.93</td>
<td>2.67</td>
<td>0.05</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Some processor names can contain more than 70 characters. In order to limit the processor name length in the results table, the full processor name is provided in a tooltip when you move the mouse over the processor name.

For compatibility and search/lookup needs, processor data tags have the following attributes:

- Title (compatibility) and pname (lookup) attribute contain full processor name.
- alt contains extended processor information for tooltip (html)
2.6 Sitecore Debugger

Sitecore provides a browser-based debugger which helps to locate problems associated with faulty or slow presentation components. The debugger helps developers analyze the output of both an entire page and the individual presentation components on a page.

2.6.1 Procedure to activate Sitecore Debugger

This procedure requires that:

- The Sitecore user has read access to the following item in the Core database:
  /sitecore/content/Documents and settings/All users/Start menu/Right/Debug
- The allowDebug attribute for the current <site> is set to true in the web.config file — or in the config include.
- The enableDebugger attribute for the current <site> is set to true in the web.config file — or in the config include.

To activate the Sitecore Debugger:

1. Log in to the Sitecore Desktop.
2. Click Sitecore, and then click Debug

A new browser window or tab opens displaying the home page of the current website with the debugging ribbon across the top. You can use the button in the far right top of the ribbon to show or hide the debugging ribbon.
2.6.2 Usage

In the Sitecore debugging ribbon, in the Mode group, click Edit to enable or disable inline editing. You can also switch between Preview and Debug mode.

In the Profile group, you can activate or de-activate Sitecore profiling for the current page. If Sitecore profiling is activated, you can save or download a profile report for the current page.

A profile report is a high-level summary of the performance of a page. You can use the profile report to identify underperforming components within a page, and underperforming steps within those components.

When Sitecore profiling is activated, a profile report appears at the bottom of the page. The profile report attempts to identify hot spots — the worst-performing components, and components that accessed a large number of items. The profile report may contain error messages, such as, if layout details indicate to bind a presentation component to a placeholder that does not exist — which
otherwise generates no output to the page.

In the Rendering group, you can enable and disable the Borders and Information options. Enabling the Borders option displays borders that outline each presentation component on the current page. Enabling the Information option displays information icons — green triangles — for each presentation component on the current page.
Holding the mouse over an information icon displays information about that individual presentation component inline in the page, including its profile, cache settings and output.

![Information Icon](image)

**Note**
When you select the Information checkbox, Sitecore invokes each presentation component on each page request and never retrieves the output of a presentation component from cache. To debug presentation component caching, do not select the Information checkbox.

In the **Trace** group, you can activate or de-activate Sitecore tracing for the current page. If Sitecore tracing is enabled, you can save or download a trace report for the current page. A trace is a low-level description of the steps involved in the page generation process.

![Trace Group](image)

If Sitecore tracing is activated, the trace report appears at the bottom of the page. You can use the trace report to determine whether or not Sitecore retrieved the output of each presentation component from cache as well as the order in which components were added to the page and the time needed to
render each component.

<table>
<thead>
<tr>
<th>Type</th>
<th>Action</th>
<th>Elapsed since last entry</th>
<th>Elapsed since start</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting trace</td>
<td></td>
<td>0.00 ms</td>
<td>0.00 ms</td>
</tr>
<tr>
<td>Current site is</td>
<td>&quot;website&quot;</td>
<td>0.01 ms</td>
<td>0.01 ms</td>
</tr>
<tr>
<td>Current domain is</td>
<td>&quot;extranet&quot;</td>
<td>0.01 ms</td>
<td>0.02 ms</td>
</tr>
<tr>
<td>Current language is</td>
<td>&quot;en&quot;</td>
<td>0.02 ms</td>
<td>0.05 ms</td>
</tr>
<tr>
<td>Profiling is active.</td>
<td></td>
<td>0.03 ms</td>
<td>0.05 ms</td>
</tr>
<tr>
<td>Device set to</td>
<td>&quot;Default&quot;</td>
<td>2.33 ms</td>
<td>2.36 ms</td>
</tr>
<tr>
<td>Current item is</td>
<td>&quot;/sitecore/content/home/&quot;</td>
<td>4.00 ms</td>
<td>6.07 ms</td>
</tr>
<tr>
<td>Current layout is</td>
<td>&quot;Sample Layout&quot;</td>
<td>5.64 ms</td>
<td>7.51 ms</td>
</tr>
<tr>
<td>Using physical layout</td>
<td>&quot;/Layouts/Sample Layout.aspx&quot;</td>
<td>7.03 ms</td>
<td>7.03 ms</td>
</tr>
<tr>
<td>Checking security for current user</td>
<td>&quot;Sitecore\admin&quot;</td>
<td>2.00 ms</td>
<td>3.92 ms</td>
</tr>
<tr>
<td>Access granted as the current user</td>
<td>&quot;Sitecore\admin&quot; has read access to current item.</td>
<td>9.65 ms</td>
<td>10.00 ms</td>
</tr>
<tr>
<td>Inserting rendering</td>
<td>&quot;Sample Sublayout&quot; into page.</td>
<td>1.25 ms</td>
<td>10.00 ms</td>
</tr>
<tr>
<td>Inserting rendering</td>
<td>&quot;Sample Inner Sublayout&quot; into page.</td>
<td>10.92 ms</td>
<td>10.92 ms</td>
</tr>
<tr>
<td>Inserting rendering</td>
<td>&quot;Sample Rendering&quot; into page.</td>
<td>10.03 ms</td>
<td>10.03 ms</td>
</tr>
<tr>
<td>Adding H1p headers to disable caching.</td>
<td></td>
<td>13.54 ms</td>
<td>13.54 ms</td>
</tr>
<tr>
<td>Adding H1p header to indicate last modification.</td>
<td></td>
<td>13.54 ms</td>
<td>13.54 ms</td>
</tr>
<tr>
<td>Starting rendering</td>
<td>&quot;(VisitorIdentification)&quot;</td>
<td>1.80 ms</td>
<td>15.46 ms</td>
</tr>
<tr>
<td>Elapsed time since start render start: 0.042 ms.</td>
<td></td>
<td>6.65 ms</td>
<td>25.10 ms</td>
</tr>
<tr>
<td>Collected debug information</td>
<td></td>
<td>2.82 ms</td>
<td>25.10 ms</td>
</tr>
<tr>
<td>Finished rendering</td>
<td>&quot;(VisitorIdentification)&quot;</td>
<td>4.61 ms</td>
<td>29.23 ms</td>
</tr>
<tr>
<td>Starting rendering</td>
<td>&quot;(WebEditRibbon)&quot;</td>
<td>0.11 ms</td>
<td>29.34 ms</td>
</tr>
<tr>
<td>Elapsed time since start render start: 150.586 ms.</td>
<td></td>
<td>150.58 ms</td>
<td>179.94 ms</td>
</tr>
<tr>
<td>Finished rendering</td>
<td>&quot;(WebEditRibbon)&quot;</td>
<td>0.04 ms</td>
<td>179.98 ms</td>
</tr>
<tr>
<td>Starting rendering</td>
<td>&quot;Placeholder main&quot;</td>
<td>0.01 ms</td>
<td>179.99 ms</td>
</tr>
<tr>
<td>Starting rendering</td>
<td>&quot;Sublayout: /Layouts/Sample Sublayout.ascx&quot;</td>
<td>1.83 ms</td>
<td>181.82 ms</td>
</tr>
<tr>
<td>Starting rendering</td>
<td>&quot;Placeholder: centercolumn&quot;</td>
<td>1.55 ms</td>
<td>183.37 ms</td>
</tr>
<tr>
<td>Starting rendering</td>
<td>&quot;Sublayout: /Layouts/Sample Inner Sublayout.ascx&quot;</td>
<td>1.61 ms</td>
<td>184.98 ms</td>
</tr>
<tr>
<td>Starting rendering</td>
<td>&quot;Placeholder: banner&quot;</td>
<td>1.49 ms</td>
<td>186.46 ms</td>
</tr>
<tr>
<td>Elapsed time since start render start: 0.015 ms.</td>
<td></td>
<td>1.89 ms</td>
<td>186.51 ms</td>
</tr>
<tr>
<td>Collected debug information</td>
<td></td>
<td>0.07 ms</td>
<td>186.58 ms</td>
</tr>
</tbody>
</table>
Chapter 3

Search Diagnostics

The tools and procedures described in this chapter are intended to help identify, troubleshoot, and resolve Sitecore search-related issues. Utilizing the outlined techniques can also provide you with actionable information related to improving search performance.

This chapter contains the following sections:

- FillDB
- Verbose Logging
- LinqScratchPad
3.1 FillDB

FillDB is a tool that allows Sitecore developers to rapidly fill databases with sample data.

The procedures described in this section are complementary to CMS tuning and diagnostics. By tuning the Sitecore CMS before and after data generation and running the diagnostic procedures before and after data generation, performance metrics can be compared across different-sized data sets. Developers and system administrators can use these metrics to identify potential bottlenecks in solution or system architecture.

3.1.1 Generating Items Using the FillDB Page

This perform this procedure, you must have permission to browse the aspx pages in the http://<site>/sitecore/admin folder.

Important
Sitecore does not recommend using the FillDB page to create sample items in any production database. You should use a test database or temporary database for sample items.

The sample items generated by the FillDB page are based on the Sample Item template that is found in standard Sitecore installations and also make use of the Title and Text fields in that template. The default Folder template is also used to create the hierarchy in which the sample items are stored. It is important that these templates exist in your Sitecore instance prior to using the FillDB process.

To use FillDB:

1. Locate the ItemGenerator.sql file in the /sitecore/admin/SqlScripts directory of your website.
2. Execute the SQL script in this file against the Sitecore Master database into which you’d like to insert sample items.
   Create a directory called data in your website root directory and then create a directory within the data directory named words — c:\sitename\Website\data\words.
3. In the data\words directory, place some .txt files that contain large bodies of text.
   Note
   Project Gutenberg provides a convenient resource for downloading free text-based versions of books - http://www.gutenberg.org/
5. In the Parent Guid field, enter the ID of the item in your content tree under which you’d like to create the sample items. The default value provided in the field belongs to the standard Sitecore home item - /sitecore/content/Home.
   Note
   The sample items generated by the FillDB page are placed in an auto-generated hierarchy of folders and items with no more than a 100 items in any given branch.
6. In the Database Name field, enter the name of the Sitecore database into which you’d like to insert sample items. The name should match the id attribute of a database that is defined in the databases section of the web.config file. The default value for this field is master.
7. In the Number of items field, specify the number of sample items that you want to create.
8. Click Go! To create the items.
   After the sample items have been created, you see a message indicating the time it took to create the items.
9. When the items have been created, clear your website cache.

To clear the website cache:

- **Browse to the Sitecore admin cache page:**
  
  http://<site>/sitecore/admin/cache.aspx
  
  On this page, click **Clear all** to clear all the Sitecore caches.

- **Reset IIS.**

- **Recycle application pool for this website.**
3.2 Verbose Logging

Verbose search logging is designed to facilitate search index configuration and provide insight in search troubleshooting scenarios. For example, if a particular item is not getting indexed, the VerboseLogger can provide more context and help you resolve the problem.

**Important**

Only enable the VerboseLogger component in special circumstances and never run it for long periods in a production environment. Otherwise, this would result in an extremely large log file, which may have performance implications.

To enable verbose search logging

1. Edit the `App_Config/Include/Sitecore.ContentSearch.config` file
2. Set the `ContentSearch.EnableSearchDebug` setting to true:
   ```xml
   <setting name="ContentSearch.EnableSearchDebug" value="true" />
   ```
   Alternatively, in the `web.config` file, add or patch the `ContentSearch.VerboseLogging` setting:
   ```xml
   <setting name="ContentSearch.VerboseLogging" value="true" />
   ```
4. In the `web.config` file, for the Sitecore search log, set the `log4net` logging level value to `DEBUG`:
   ```xml
   <log4net>
   <logger name="Sitecore.Diagnostics.Search" additivity="false">
   <level value="DEBUG" />
   <appender-ref ref="SearchLogFileAppender" />
   </logger>
   </log4net>
   ```
3.3 LinqScratchPad

Sitecore 7.0 includes a LINQ search API which provides convenient index searching capabilities using standard LINQ queries and syntax. LinqScratchPad is a tool that gives Sitecore developers the ability to execute LINQ search queries quickly and easily for testing and evaluation purposes.

3.3.1 Required Skills

- A working knowledge of Sitecore search queries and LINQ syntax.

3.3.2 Usage

To use the LinqScratchPad tool you must have permission to browse to the aspx pages in the http://<site>/sitecore/admin directory.

Browse to the LinqScratchPad page:
http://<site>/sitecore/admin/LinqScratchPad.aspx

You are presented with an interface like this:

This interface allows you to write C# code that is compiled and executed. However, to ensure that the code executes properly for this page, you must follow a few conventions:

- Use the Test.Program type — namespace Test, class Program
The Program class must also contain two methods: **Main** and **RunTimer**.

- **The Test.Program.Main method** is used to execute the query/code you are testing. The Test.Program.Main method should accept one argument of type string and return a value of type IEnumerable<SearchResultItem>.

- **The Test.Program.RunTimer method** is used to report the amount of time spent in executing your code from the Test.Program.Main method. The Test.Program.RunTimer method should accept one argument of type string and return a value of type string.

You will also notice a private Stopwatch field named stopWatch. This field can be used to measure the execution time of your query/code in the Test.Program.Main method, then used to report execution time via the Test.Program.RunTimer method. This field is not required, however, for proper execution.

### Important

The Test.Program.Main method returns a value of type IEnumerable<SearchResultItem> that is used to populate a grid of results on the LinqScratchPad page. It is important to note that returning a large result set, for example, thousands of results, will cause the page to render results slowly and may not be helpful when evaluating query execution time. It is recommended that you limit the number of results returned to avoid unnecessary delay.

For more information about using the LINQ to Sitecore search API, see the manual *Developer’s Guide to Item Buckets and Search* on the SDN.