The Dispatcher

How to Set Up and Use FileNet's Caching and Load Balancing Tool.

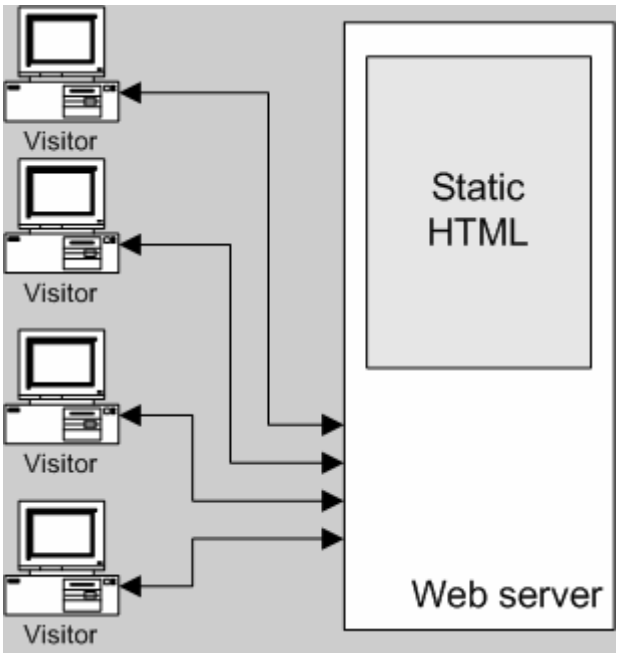
1 Purpose

The Dispatcher is FileNet's caching/load balancing tool. This section tells you how to use the dispatcher efficiently, and how to design your Web site so that the dispatcher can handle it best. Read this document before you create a high-performance Web site, and use it to fine-tune your dispatcher settings should performance issues arise.

2 How the Dispatcher Works

There are two basic approaches to Web publishing. Standard HTTP servers, such as Apache or IIS, are very simple, but fast. If you want dynamic, real-time, intelligent Enterprise Content Management, you have to pay a price in terms of the additional computation time needed. The dispatcher helps you to have an environment that is both fast and dynamic, and this section tells you how this works.

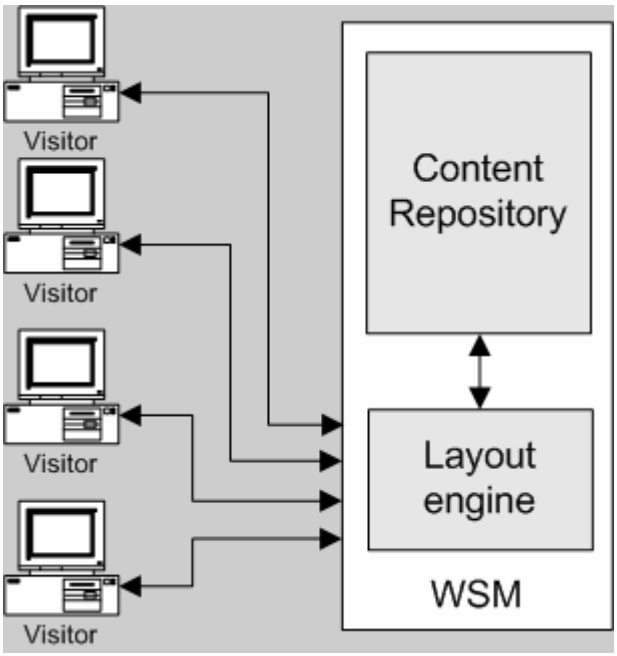
2.1 Static Web Server



A static Web server, such as Apache or IIS, serves pre-built, static HTML files to visitors of your Web site. This process is very simple, and thus extremely efficient. If a visitor requests a file (say, an HTML page), the file is usually taken directly from memory, or at worst it is read from the local drive.

Because static Web servers have been around for quite some time, they offer a lot of tools for administration and security management, and they are very well integrated with network infrastructures.

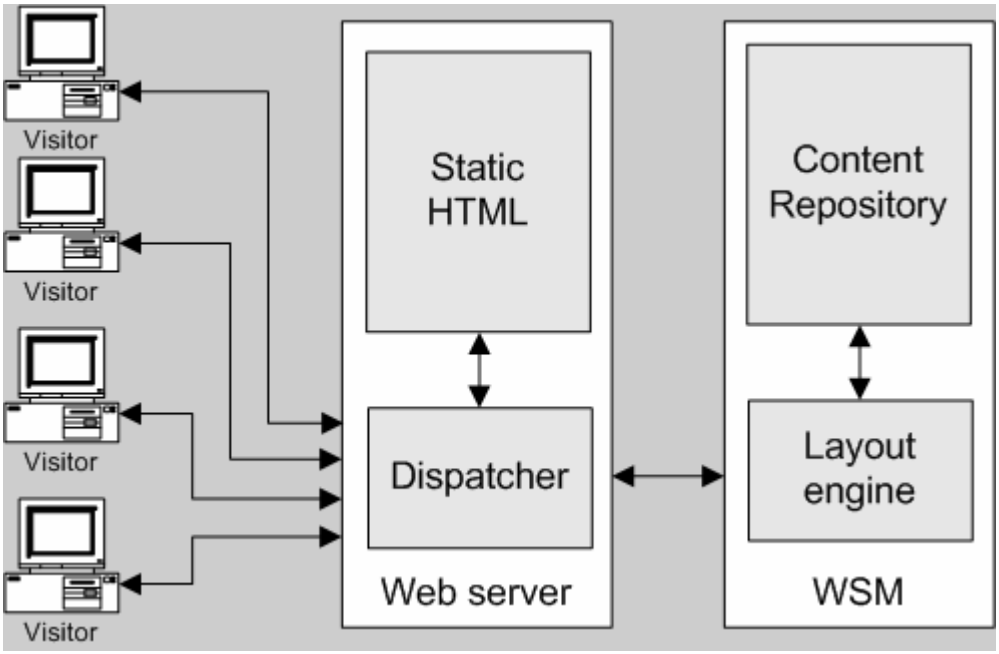
2.2 Content Management Server



If you use a Content Management Server, such as FileNet WSM, an advanced layout engine processes the request from a visitor. The engine reads content from a repository and uses styles, formats and access rights to transform the content into a document that is tailored to the visitor's needs and rights.

This allows you to create richer and more dynamic content, which is easier to manage and requires less maintenance work. However, the layout engine requires more processing power than a static server, so this setup may be prone to slowdown if many visitors use the system.

2.3 The Dispatcher and Caching



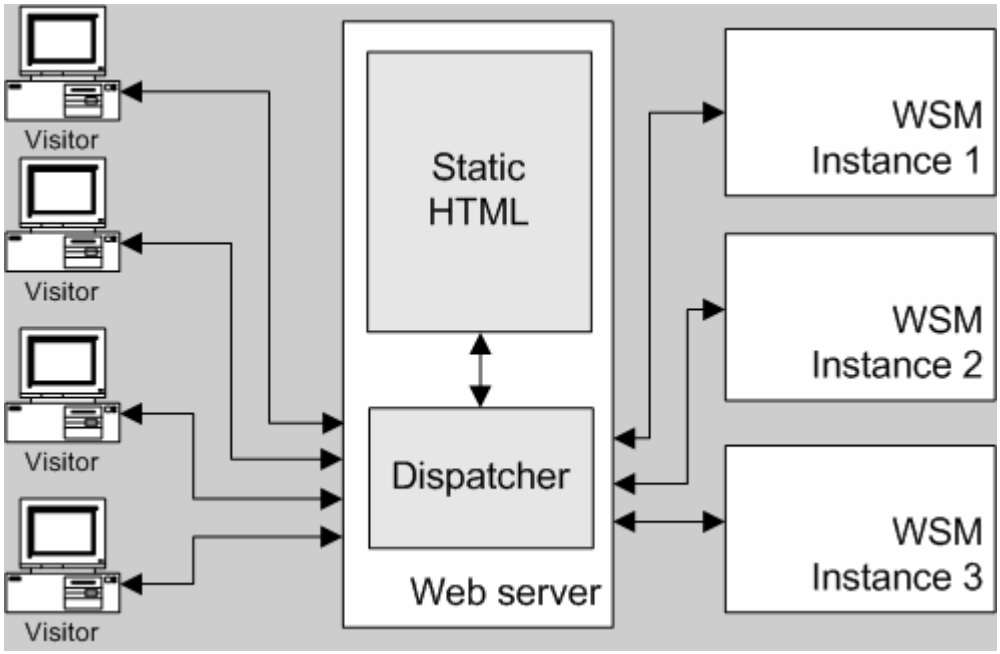
The dispatcher works as part of a static HTML server, such as Apache. The aim of the dispatcher is to store (or “cache”) the Site content as much as possible in the form of a static Web site, and access the layout engine as little as possible. This approach brings you the best of two worlds:

- Static content is handled with exactly the same speed and ease as on a static Web server.
- You can have as much dynamic content as needed, without slowing the system down any more than absolutely necessary.

You even get a few bonuses for free: You can use all the administration and security tools for your static Web servers, and you can manage the static parts of your Web site with the same ease as the dynamic ones.

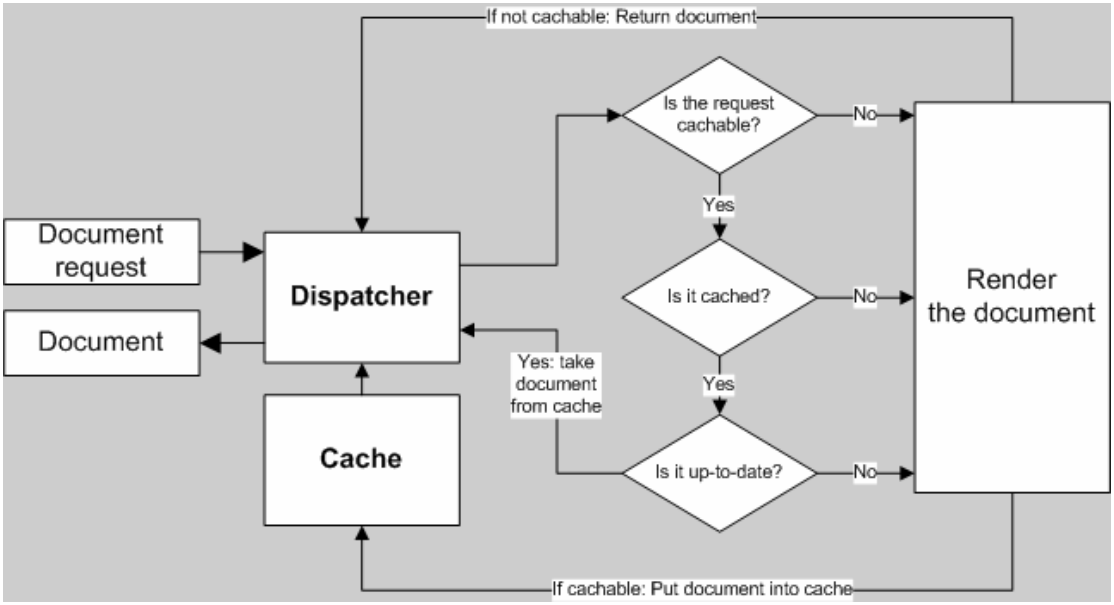
The dispatcher contains mechanisms to keep the static HTML up-to-date if you change the content of the dynamic site, and you can specify in detail which documents are stored as static files and which are always computed dynamically.

2.4 The Dispatcher and Load Balancing



But the dispatcher does even more: You can split up the dynamic parts to different servers, so you gain additional processing power, the server is fail-safe, or you can manage different Web sites on the same static Web server. All of these techniques are summarized as “load balancing”. It means that servers can share their work seamlessly. The dispatcher allows for a number of very useful load balancing scenarios.

3 Returning Documents



Finding out if a Document is Cacheable

You can define which documents the dispatcher caches in the configuration file. The dispatcher checks the request against the list of cacheable documents. If the document is not cacheable, the dispatcher requests the document from the WSM instance.

The dispatcher always requests the document from the WSM instance in the following cases:

- If the HTTP method is not GET. Other common methods are POST for form data and HEAD for the HTTP header.
- If the request URI contains a question mark (“?”). This usually indicates a dynamic page, such as a search result, which does not need to be cached.
- The file extension is missing. The Web server needs the extension to determine the document type (the MIME-type).
- The authentication header is set (this can be configured)

Finding out if a Document is Cached

The dispatcher stores the cached files on the Web server as if they were part of a static Web site. If a user requests a cacheable document, the dispatcher checks if that document exists in the Web

server's file system. If not, the dispatcher requests the document from the WSM instance.

Finding out if a Document is Up-To-Date

To find out if a document is up to date, the dispatcher performs two steps:

- It checks whether the document is subject to auto-invalidation. If it is not, the document is considered up-to-date.
- If the document may auto-invalidate, the dispatcher checks if it is older or newer than the last change on the Web site. If it is older, the dispatcher requests the current version from the WSM instance and replaces the version in the cache with the new one.

Note that documents without auto-invalidation remain in the cache until they are physically deleted, for example by a content update on the Web site.

4 Caching

The dispatcher has two primary methods for updating the cache content when the Web site changes. Content updates remove precisely the pages that have changed, as well as files that are directly associated with them. The Auto-Invalidation process automatically and liberally invalidates those parts of the cache that may be out of date after an update.

4.1 Content Updates

In a content update, one or several WSM documents change. WSM sends a syndication request to the dispatcher, which updates the cache accordingly:

- It deletes the modified files from the cache.
- It deletes all files that start with the same handle from the cache. For example, if the file en/index.html is updated, all the files that start with “en/index.” are deleted. This mechanism allows you to design cache-efficient sites, especially in regard to picture navigations.
- It touches the so-called stat file. The timestamp of the stat file now contains the date of the last change.

The files that are affected by a content update are removed, but not replaced right away. The next time such a file is requested, the dispatcher fetches the new file from the WSM instance and puts it into the cache.

Typically, automatically generated pictures that use text from a page are stored in picture files that start with the same handle. For example, you may store the title text of the page mypage.html as the picture mypage.titlePicture.gif in the same folder, so the picture is deleted from the cache automatically each time the page is updated. This way, you can be sure that the picture always reflects the current version of the page.

Note: You may have several stat files, for example one per language folder. If a page is updated, WSM looks for the next parent folder that has a stat file, and touches that file.

4.2 Auto-Invalidation

Auto-invalidation automatically invalidates large parts of the cache, without physically deleting any files.

- At every content update, the so-called stat file is touched, so its timestamp reflects the last content update.
- The dispatcher has a list of files that are subject to auto-invalidate. When a document from that list is requested, the

dispatcher compares the date of the cached document with the timestamp of the stat file. If the cached document is newer, the dispatcher returns it. If it is older, the dispatcher fetches the current version from the WSM instance.

Note that auto-invalidation does not involve any action by the dispatcher at update time, except for touching the stat file. However, touching the stat file automatically renders the cache content obsolete, without physically removing it from the cache.

Auto invalidation is typically used for HTML pages. Because these pages contain links and navigation entries, they usually have to be updated after a content update. If you have automatically generated PDF or picture files, you may choose to auto-invalidate those too.

4.3 The Cache Directory

For caching, the dispatcher module uses the Web server's ability to serve static content. The dispatcher places the cached documents in the document root of the Web server.

Because of this process, the dispatcher stores only the HTML code of the page, but does not store the HTTP headers. This can be an issue if you use different encodings within your Web site, which may get lost.

5 Load Balancing

Load balancing means that you can spread the computational load of the Web site across several Web servers. In practice, the dispatcher sends document requests to several Web servers. Because each server has fewer documents to process, you have faster response times. The dispatcher keeps internal statistics for each document category, so it can estimate the server load and distribute the queries efficiently.

In addition to faster load times, load balancing can be used for fail safing. If the dispatcher does not receive responses from a Web server, it will automatically relay requests to the other server or servers. Thus, if a server becomes unavailable, the only thing that happens is that the site slows down according to the computation power that is lost. However, all services will continue as usual while the problem is solved.

Note: While load balancing spreads the load efficiently, caching helps to reduce the load. Therefore, try to optimize caching and reduce the overall load before you set up load balancing. Good caching may increase the load balancer's performance, or render load balancing unnecessary.

5.1 Document Categories

The dispatcher keeps an internal statistics about how fast the Web servers process documents. Based on this data, the dispatcher estimates which server will return a request quickest, and it blocks the necessary computation time for that server.

Because different types of requests may take different times to complete, the dispatcher allows you to specify document categories. For example, you may make a distinction between HTML pages and images. This mirrors that HTML documents may have different typical response times than images, which the dispatcher considers when computing the time estimates.

If you use an elaborate search function, you may create a new category for search queries. This helps the dispatcher to send search queries to the server that answers them fastest, and it prevents a slower server from stalling because it receives several "expensive" search queries while the other servers get the "cheaper" requests.

5.2 Sticky Connections

Sticky connections ensure that the documents for one user are all composed on the same server. This is important if you use personalized pages and session data. This data is stored on the

Web server, so subsequent requests from the same user must go to the same server, or the data is lost.

Because sticky connections restrict the dispatcher's ability to optimize the requests, you should use them only when needed. You can specify the folder that contains the "sticky" documents. All the documents in the folder are composed on the same server for each user.

Note: For most pages that use sticky connections, you have to switch off caching. If you use caching, the page looks the same to all users, regardless of the session content. For some applications, it can be possible to use both sticky connections and caching, for example if you display a form that writes data to the session.

---

6 Optimizing a Web Site for Caching

WSM's cache system offers a number of built-in mechanisms that you can use if your Web site takes advantage of them. This section tells you how to design your Web site so that it will benefit the most from caching.

It may help you to remember that the dispatcher stores the cache on a standard Web server. This means that you can cache everything that you can store as a page and request using an URL. You cannot store other things, such as HTTP headers, cookies, session data and form data. In general, a lot of caching strategies involve selecting good URLs and not relying on this additional data.

6.1 Page Encoding

HTTP headers are not cached, which may be an issue if you store the encoding in them. Taken from the cache, the page will have the default encoding for the Web server. There are two ways of solving this:

- If you use only one encoding, make sure that the encoding you use on the Web server is the same as the default encoding of the WSM Web site.
- Use a <META> tag in the HTML header to set the encoding, such as:
  <META http-equiv="Content-Type" content="text/html; charset=EUC-JP">

6.2 Avoid URL Parameters

If possible, avoid URL parameters for pages that you want to cache. For example, if you have a picture gallery, the following URL is never cached:

```
www.myCompany.com/pictures/gallery.html?event=christmas&page=1
```

However, you can put these parameters into the page URL, as follows:

```
www.myCompany.com/pictures/gallery.christmas.1.html
```

Note that this URL calls the same page and the same template as gallery.html. In the template definition, you can specify which script renders the page, or you can use the same script for all pages.

6.3 Customize by URL

If you allow users to change the font size (or do any other customization on the layout), make sure that the different layouts are reflected in the URL as well. If you store them, say, in a cookie, then it is pure coincidence which version is cached. As a result, the

dispatcher will return documents of all font sizes at random. Instead, make the font size part of the URL:

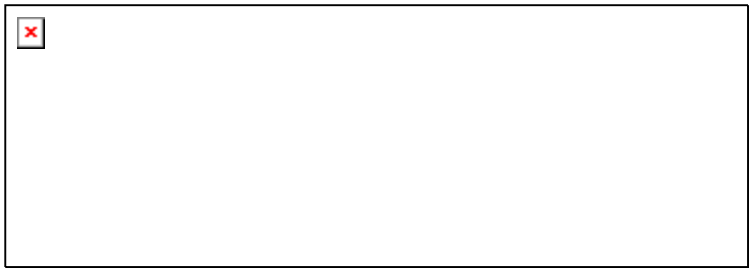
```
www.myCompany.com/news/main.large.html
```

Note: For most layout aspects, it is also possible to use style sheets and/or client side scripts. These will usually work very well with caching. This is also useful for a print version. Use an URL as follows for the print version:

```
www.myCompany.com/news/main.print.html
```

Using the script globbing of the template definition, you can specify a separate script that renders the print pages.

6.4 Picture Titles



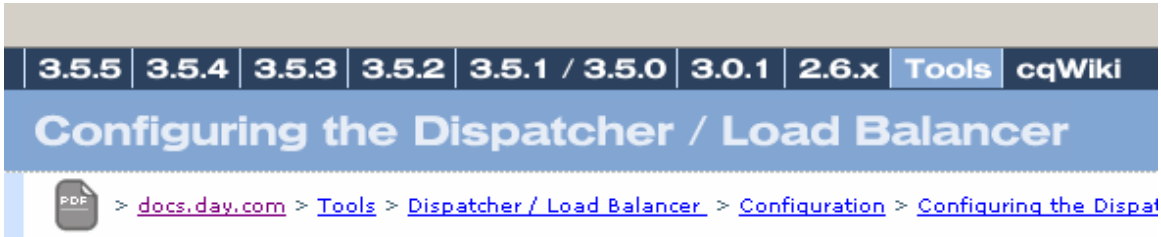
If you render page titles or other text as pictures, store the files so that they are deleted at a content update for the page. To do so,

- place the picture file in the same folder as the page.
- use the same name as the filename of the page, plus a dot, plus the picture name.

For example, you can store the title of the page myPage.html in the file myPage.title.gif. This file is automatically deleted if the page is updated, so any change to the page title is automatically reflected in the cache.

Note that the file does not have to physically exist on the WSM instance. You can use a script that dynamically creates and outputs the picture. The dispatcher then stores the picture on the Web server.

6.5 Picture Navigation



If you use pictures for the navigation entries, the method is the same as above, but slightly more complex. The trick is that all the navigation images are stored with the target pages. If you use two pictures for normal and active, you can use the following scripts:

- A script that displays the page, as normal.
- A script that processes ".normal" requests and returns the normal picture.
- A script that processes ".active" requests and returns the activated picture.

It is important that you create these pictures on the same template as the page. Like this, a content update (for example, if you change the name of the page) will delete these pictures as well as the page. For pages that are not modified, the pictures still remain in the cache, although the pages themselves are usually auto-invalidated.

6.6 Personalization

Because the dispatcher cannot cache personalized data, limit personalization to where it is necessary. For example, if you use a freely customizable start page, that page has to be composed every time a user requests it. If, in contrast, you offer a choice of 10 different start pages, you can cache each one of them, improving performance.

Note: If you personalize each page, for example by putting the user's name into the title bar, you cannot cache it, which can cause a major performance impact. If you have to do this, use iFrames to split the page into one part that is the same for all users and one part that is the same for all pages of the user. You can then cache both of these parts. Alternatively, you can use client-side JavaScript to display personalized information. However, you may have to make sure that the page still displays correctly if a user turns JavaScript off.

6.7 Sticky Connections

You can define one folder with sticky documents for each Web site. Group all the documents that require sticky connections in one folder, and try not to have other documents in it. If a user leaves this folder and later returns to it, the connection still sticks.

6.8 MIME-Types

There are two ways in which a browser can determine the type of a file:

- By its extension (such as “.html”)
- By the MIME-type that the server sends along with the file.

For most files, the MIME-type is implied in the file extension. For example, files ending in “.html” have the MIME-type “text/html”, and files ending in “.jpg” have the MIME-type “image/jpeg”. If the file has no ending, it is displayed as plain text.

The MIME-type is part of the HTTP header, and as such, the dispatcher does not cache it. If your WSM application returns files that do not have a recognized file ending, but rely on the MIME-type instead, these files may be displayed wrongly.

To make sure that files are cached properly, follow these guidelines:

- Make sure that files always have the proper extension.
- Avoid generic file serve scripts, which have URLs like “download.jsp?file=2214”. Re-write the script to use URLs like “download.2214.pdf”.

7 Installation

To use the dispatcher, you have to set it up, and connect WSM to it, so that the dispatcher correctly updates the cache if WSM content changes.

7.1 Dispatcher

7.1.1 Installing

The basic installation procedure for the dispatcher is the same for all platforms:

- Install the Web server (IIS, Apache, Sun).
- Install the Dispatcher.
- Configure the Web server configuration file.

The WSM installer automatically installs the dispatcher source files in the folder **\opt\dispatchers** of the WSM installation directory.

Each dispatcher version comes as an archive file, such as:

```
dispatcher_apache_5227_linux2.4.tgz
```

This archive contains the dispatcher release number 5227 for an Apache Web server that runs under Linux 2.4.

Each archive contains the following files:

- The dispatcher.
- An example configuration file.
- A readme file with detailed installation instructions.

Because of the differences between the supported Web servers, the files may differ significantly, and additional files may be available. For more information, refer to the readme file. For example, in the above file, the readme file is in the folder /conf and named as follows:

```
README.dispatcher.apache
```

7.1.2 Protecting from Unauthorized Flush Requests

7.1.3 Logging

In the Web server configuration, you can set the dispatcher log file and log level. Refer to the readme file of your dispatcher instance for more information.

Note that by default, the log level is high, so the dispatcher logs all errors and warnings. If the dispatcher works smoothly, you can lower the log level.

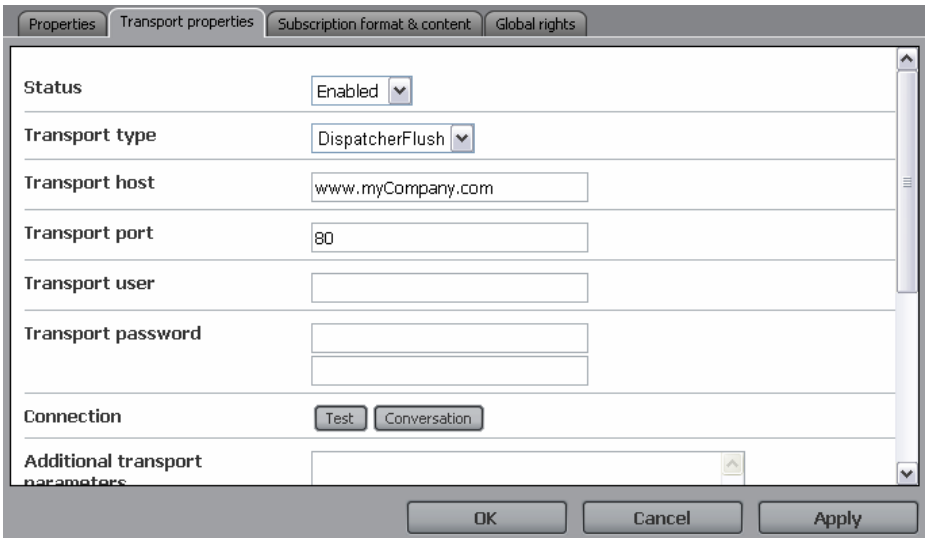
7.2 WSM

7.2.1 Invalidating from the Authoring Environment

In this set up, the WSM authoring environment sends a cache invalidation request to the dispatcher when it publishes a page. This removes the old page content from the cache when new content is published.

To set up WSM's authoring environment so that it invalidates the cache upon publication of a page, proceed as follows:

1. Log in to WSM's authoring and administration environment.
2. Click the **Users** tab. Click the **Replication Users** folder, and then click the **Dispatcher Cache Flushing Agent**. WSM opens the configuration window.
3. Click the **Transport Properties** tab. WSM displays the transport properties.
4. In the Status list, click **Enabled**. Type the host name and port of the Web server where the dispatcher runs. Click **OK** to activate the agent.



Note: The cache flushing agent does not require a user name and password.

There are two potential issues with this approach:

- The dispatcher must be reachable from the authoring instance. If your Web server has severely restricted access to the network, this may not be the case.
- Publication and cache invalidation take place at the same time. In some cases, a user may request a page just after it was removed from the cache, and just before the new page is published. WSM now returns the old page, and the dispatcher caches it again. This is more of an issue for large sites.

7.2.2 Invalidating from the Publishing Environment

To invalidate the cache from the publishing environment, you need to modify

7.2.3 Using Multiple Dispatchers

In complex setups, you may use multiple dispatchers. For example, you may use one dispatcher to publish a Web site on the Intranet, and a second one, under a different address and with different security settings, to publish the same content on the Internet.

In this case, make sure that each request goes through only one dispatcher. The dispatcher does not handle requests that come from an other dispatcher.

For the above case, make sure that both dispatchers access the WSM Web site directly.

8 Dispatcher Configuration

By default, the dispatcher configuration is stored in the file dispatcher.any. You may change the name and location of this file during installation.

The file has the following general structure:

```
/name "internet-server"
 /farms
  {
  # List of Web sites that are handled by the dispatcher
  }
```

One dispatcher can cover a number of Web sites, such as intranet.myCompany.com, internet.myCompany.com and www.myFlagshipProduct.com. For each Web site, you can specify separate caching and rendering parameters. The configuration for each Web site looks as follows (note that the first header, myCompany, is the name of the section):

```
  /myCompany
   {
   /clientheaders
    {
    # List of headers that are passed on
    }
   /virtualhosts
    {
    # List of URLs for this Web site
    }
   /sessionmanagement
    {
    # settings for user authentification
    }
   /renders
    {
    # List of WSM instances that render the documents
    }
   /filter
    {
    # List of filters
    }
   /cache
    {
    # Cache configuration
    /rules
     {
     # List of cachable documents
     }
    /invalidate
     {
     # List of auto-invalidated documents
     }
    }
   /statistics
    {
    /categories
```

```
     {
     # The document categories that are used for load
balancing estimates
     }
    }
   /stickyConnectionsFor "/myFolder"
   }
```

8.1 General Settings

For a farm, such as myCompany, you can specify the following parameters:

/homepage (optional)

The page that the dispatcher returns when no target page or file is requested. In a typical setup, this is the page that is returned when a user requests “www.myCompany.com”. The /homepage parameter is required if there is no automatic redirecting from the server (for example, for IIS) or from WSM (for example, if you shut it down after the content is cached).

To display the index.html page, use the following setting:

```
    /homepage "/index.html"
```

/propagateSyndPost (optional)

By default, the dispatcher does not pass syndication requests on to WSM. If set to 1, the dispatcher passes the syndication requests on:

```
    /propagateSyndPost "1"
```

Note: When you enable this feature, make sure that you do not filter POST requests.

8.2 Configuring the Virtual Hosts

The virtual hosts section is a list of all the hostnames/URI combinations that the dispatcher accepts for this Web site. For example, the section

```
   /virtualhosts
    {
    "www.myCompany.com"
    "www.myCompany.ch"
    }
```

handles the requests for both the .com and the .ch address. Note that you can use the asterisk ("*") character, so you could also write

```
   /virtualhosts
```

```
    {
    "www.myCompany.*"
    }
```

to handle the requests for all myCompany domains. If you want the dispatcher to handle all requests that come its way, you can use

```
   /virtualhosts
    {
    "*"
    }
```

If you use more than one render farm, the dispatcher always uses the first farm where the request matches the virtual host list. In the following example, all the pages in the “product” folder are sent to server 1, all other pages to server 2. Note that for clarity, the example contains only the sections that are relevant to this discussion.

```
/farms
  {
  /myProducts
    {
    /virtualhosts
      {
      "www.mycompany.com/products/*"
      }
    /renders
      {
      /hostname "server1.myCompany.com"
      /port "80"
      }
    }
  /myCompany
    {
    /virtualhosts
      {
      "www.mycompany.com"
      }
    /renders
      {
      /hostname "server2.myCompany.com"
      /port "80"
      }
    }
  }
```

8.3 Configuring Session Management and Authentification

This feature allows you to create a secure session for access to the render farm. Users need to log in before they can access any page in the farm, and have access to all pages after logging in. If you have separate sections of your Web site with different access, you need to specify multiple render farms.

***/*directory** (mandatory)

The directory that stores the session information. If the directory does not exist, it is created.

/encode (optional)

How the session information is encoded. Use “md5” for an encryption using the md5 algorithm, or “hex” for hexadecimal encoding. If you encrypt the session data, a user with access to the file system cannot read the session contents. The default is md5.

/header (optional)

The name of the HTTP header or cookie that stores the authorization information. If you store the information in the http header, use “HTTP:” and the name if the header. To store the information in a cookie, use “Cookie:” and the name of the cookie. If you do not specify a value, “HTTP:authorization” is used.

/timeout (optional)

The number of seconds until the session times out after it has been used last. If not specified, 800 is used, so the session times out a little over 13 minutes after the last request of the user.

A typical configuration entry may look as follows:

```
/sessionmanagement
  {
  /directory "/usr/local/apache/.sessions"
  /encode "md5"
  /header "HTTP:authorization"
  /timeout "800"
  }
```

8.4 Configuring the Client Headers

This is a list of all the HTTP headers that are passed from the client to the WSM instance. By default, the dispatcher passes the standard HTTP headers on to the WSM instance. In some instances, you may want more headers (such as custom headers) or less (such as authentication headers, which need only concern the Web server). If you need a special set of headers, you have to specify the entire set of headers that are passed on, including those that are passed on by default. Such a list might look as follows:

```
/clientheaders
   {
   "referer"
   "user-agent"
   "authorization"
   "from"
   "content-type"
   "content-length"
   "accept-charset"
   "accept-encoding"
   "accept-language"
   "accept"
```

```
   "host"
   "if-match"
   "if-none-match"
   "if-range"
   "if-unmodified-since"
   "max-forwards"
   "proxy-authorization"
   "proxy-connection"
   "range"
   "cookie"
   "cq-action"
   "cq-handle"
   "handle"
   "action"
   "cqstats"
   }
```

8.5 Configuring the Rendering WSM Instance(s)

This section configures where the dispatcher will send requests to render a document. If you use a single WSM instance for rendering, you can specify it as follows:

```
/renders
  {
    /myRenderer
      {
      /hostname "cq.myCompany.com"
      /port "80"
      }
  }
```

If the WSM instance runs on the same computer as the dispatcher, use

```
/renders
   {
    /myRenderer
     {
     /hostname "127.0.0.1"
     /port "3402"
     }
}
```

If you use multiple WSM instances, specify them as follows. The load balancer will try to spread the expected workload equally among all WSM instances.

```
/renders
  {
    /myFirstRenderer
      {
      /hostname "cq.myCompany.com"
      /port "80"
      }
    /mySecondRenderer
     {
     /hostname "127.0.0.1"
     /port "3402"
     }
```

```
}
```

8.6 Configuring Filters

Using filters, you can specify which requests the dispatcher module handles. Any requests that are not covered by the filters are sent back to the server, where they are offered to the other modules that run on the Web server. This allows you for example to exclude ASP pages from the dispatcher.

If you want the dispatcher to handle all files, use:

```
/filter
  {
  /0001
    {
    /glob "*"
    /type "allow"
    }
  }
```

The following filter does not handle ASP pages:

```
/filter
  {
  /0001
    {
    /glob "*"
    /type "allow"
    }
  /0002
    {
    /glob "*.asp *"
    /type "deny"
    }
  }
```

Note that the match is for the entire request line, not just the URI. The match is “*.asp *”, because the full line is “GET /home.asp HTTP/1.0”.

You can also match other parts of the request. The following filter does not handle form data submitted by the post method:

```
/filter
  {
  /0001
    {
    /glob "*"
    /type "allow"
    }
  /0002
    {
    /glob "POST *"
    /type "deny"
    }
  }
```

Do not confuse the filter and cache rules paragraphs. If a request is not covered by a filter, the dispatcher ignores it and it is sent to the

other Web server modules. In contrast, if a request passes the filter but is not cached, the dispatcher sends it to the WSM instance.

8.7 Configuring the Cache Parameters

The cache parameters specify some general aspects of how and where the dispatcher caches documents. The following options are supported:

/docroot

This link points to the document root of the Web server. This is where the dispatcher stores the cached documents, and this is where the Web server looks for them. If you use multiple render farms, you have to specify different document roots on the Web server for each farm, and use the corresponding link here.

/statfile

This link points to the so-called stat file, which the dispatcher uses to keep track of the last content update. This can be any file on the Web server. The file itself is empty, only the timestamp is used.

Use this if you want to use one stat file for the entire Web site.

/statfileslevel

Creates stat files in all folders down to the level you specify. Use this if you want to invalidate only parts of the cache, and not the entire cache. For a default set up with language folders (so the English site is under /content/myWebsite/en), use /statfileslevel = “2” to invalidate per language. That is, when the dispatcher invalidates the English part of the Web site, this does not affect the German part of the Web site, or other Web sites.

If you use this parameter, do not specify the /statfile parameter.

/allowAuthorized

By default, requests that carry an authentication header are not cached. This is because authentication is not checked when a document is returned from the cache, so the document may be displayed for a user who does not have the necessary rights. However, in some setups it can be possible to cache authenticated documents.

A typical cache section might look as follows:

```
/cache
  {
  /docroot "/opt/communique/dispatcher/cache"
  /statfileslevel  "2"

  /allowAuthorized "0"
```

```
  /rules
    {
    # List of files that are cached
    }
  /invalidate
    {
    # List of files that are auto-invalidated
    }
  }
```

8.8 Configuring the List of Cachable Documents

The list of cachable documents determines which documents are cached. Remember that the dispatcher never caches a document in the following instances:

- If the HTTP method is not GET. Other common methods are POST for form data and HEAD for the HTTP header.
- If the request URI contains a question mark (“?”). This usually indicates a dynamic page, such as a search result, which does not need to be cached.
- The file extension is missing. The Web server needs the extension to determine the document type (the MIME-type).
- The authentication header is set (this can be configured)

If you do not have dynamic pages beyond those already excluded by the above rules, you can let the dispatcher cache everything. The rules section for this looks as follows (note that "0000" merely denotes the first rule – you can name or number the rules in any way you like):

```
/rules
  {
  /0000
    {
    /glob "*"
    /type "allow"
    }
  }
```

If there are some sections of your page that are dynamic, such as, say, a news application and a closed user group, you can define exceptions. Note that closed user groups must not be cached because the user rights are not checked for cached pages.

```
/rules
  {
  /0000
    {
    /glob "*"
    /type "allow"
    }
  /0001
```

```
    {
    /glob "/home/news/*"
    /type "deny"
    }
  /0002
    {
    /glob "*/private/*"
    /type "deny"
    }
```

You can also compress the cached documents. This works only on Apache Web servers and allows Apache to return the document in a compressed form if the client requests this. Note that currently, only the "gzip" format is supported. The following rule caches all documents in compressed form, so Apache can return either the uncompressed or the compressed form to the client:

```
/rules
  {
  /rulelabel
    {
    /glob "*"
    /type "allow"
    /compress "gzip"
    }
  }
```

8.9 Configuring the List of Auto-Invalidate Documents

This list contains all the documents that are automatically rendered invalid after a content update. Note that the dispatcher does not physically delete the pages after the content update, but checks them for validity when they are requested. Documents in the cache that are not auto-invalidated will remain in the cache until they are deleted by a content update.

Auto-invalidate is typically used for HTML pages. Because HTML pages typically contain links to other pages and navigations, it is very hard to find out if a page is affected by a content update. To stay on the safe side, you usually auto-invalidate all HTML pages. The configuration entry for this looks as follows:

```
/invalidate
  {
  /0000
    {
    /glob "*"
    /type "deny"
    }
  /0001
    {
    /glob "*.html"
    /type "allow"
    }
  }
```

If you offer automatically generated PDF and ZIP files for download, you may have to auto-invalidate these as well. The configuration entry for this looks as follows:

```
/invalidate
  {
  /0000
    {
    /glob "*"
    /type "deny"
    }
  /0001
    {
    /glob "*.html"
    /type "allow"
    }
  /0002
    {
    /glob "*.zip"
    /type "allow"
    }
  /0003
    {
    /glob "*.pdf"
    /type "allow"
    }
  }
```

8.10 Configuring Statistics

The statistics section defines the categories that the dispatcher uses for load balancing estimates. The dispatcher keeps a list of typical response times per WSM instance per request category. When the dispatcher needs to render a document, it checks these lists to find out which WSM instance currently has the most resources to handle it, and it reserves these resources when considering future requests. The statistics are internal and there is no way to access them from outside.

By default, the dispatcher makes a distinction between HTML documents and everything else. The default statistics section looks as follows:

```
/statistics
  {
  /categories
    {
    /html
      {
      /glob "*.html"
      }
    /others
      {
      /glob "*"
      }
    }
  }
```

If you want to add a special category for search pages, the section would look as follows. Note that the more specific requests must be stated first:

```
/statistics
  {
  /categories
    {
    /search
      {
      /glob "*search.html"
      }
    /html
      {
      /glob "*.html"
      }
    /others
      {
      /glob "*"
      }
    }
  }
```

Note: If you do not use load balancing, you can omit this section.

8.11 Configuring Sticky Connections

You can define one folder that contains sticky documents. The dispatcher sends all requests from one user that are in this folder to the same Web server. This ensures that session data is present and consistent for all documents.

To define sticky connections for the folder /myFolder, use the following line:

```
   /stickyConnectionsFor "/myFolder"
```

8.12 Using Include Files

When you have a large dispatcher configuration file, you can use include files to create manageable files, or to include automatically generated files.

For example, to include the file “myFarm.any” in the farms configuration, use the following code:

```
/farms
  {
  $include "myFarm.any"
  }
```

You can use the asterisk to specify a range of files to include. For example, if the files “farm_1.any” to “farm_5.any” contain the configuration of farms one to five, you can include them as follows:

```
/farms
  {
```

```
  $include "farm_*.any"
  }
```