Escenic Content Engine

Server Administration Guide

5.7.88.187628

escenic

A CCI COMPANY

Table of Contents

1 Introduction

This **Server Administration Guide** is intended to be read by the system administrator responsible for managing the server or servers on which an Escenic Content Engine and its supporting SW components are installed. It covers the periodic administration tasks a system administrator needs to carry out once the Content Engine is installed and in operation. It does **not** describe how to install and deploy the Content Engine: for installation and deployment instructions, see the Escenic Content Engine Installation Guide.

Both this manual and the Escenic Content Engine Installation Guide make the following assumptions about the Escenic installation and you, the reader:

- The Content Engine and the supporting software stack (database, web server, application server and so on) are installed on one or more Linux servers.
- You are a suitably qualified system administrator with a working knowledge of both the operating system on which the Content Engine is installed and of the components in the supporting software stack.

All shell command examples given in the manual are tested on Debian Linux servers: they may need minor modifications to be used on other Linux platforms, and it is assumed that you are able to make the necessary "conversions" to your own platform. Some of the commands should be executed as the owner of the Escenic installation. This is signalled by use of the **$** command prompt. For example:

```
$ ls
```

Other commands must be executed as **root**. This is signalled by the use of the **#** command prompt:

```
# /etc/init.d/slapd restart
```

Two different kinds of installation are discussed in this manual:

- Single server installations, in which the Content Engine and the entire supporting SW stack are installed on a single machine.
- Multi-server installations. There are many possible multi-server configurations, but only one is described here. It is assumed that you are competent to extrapolate from the description of this configuration to your particular variant.

All file paths and URLs shown in the manual are based on the following standard folder structure:

Standard location	Component
/opt/escenic	Escenic
/opt/escenic/engine	Escenic Content Engine
/opt/escenic/assemblytool	Escenic assembly tool
/etc/escenic	Escenic configuration
/etc/escenic/engine	Escenic Content Engine configuration

Standard location	Component
`/opt/java/jdk`	Java
`/opt/java/ant`	Ant
`/opt/tomcat`	Tomcat

If your system is organized differently, then adjust the paths you use accordingly.

2 The escenic-admin Web Application

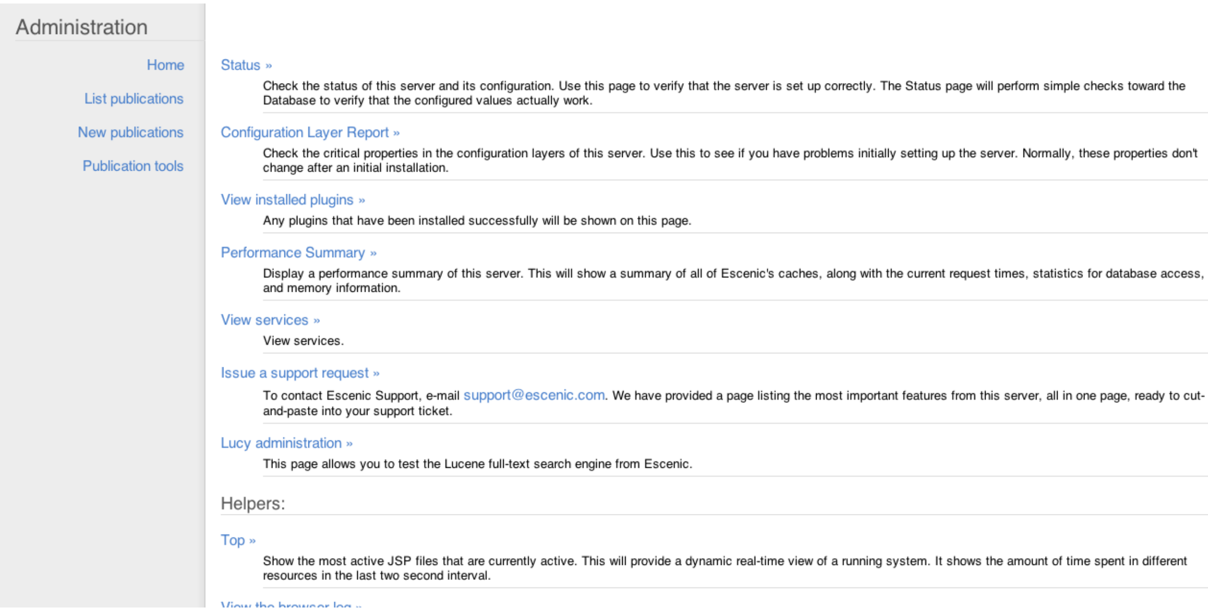
An administration web application called **escenic-admin** is included with the Escenic Content Engine. It provides access to various administration-related tools. This chapter contains a full description of **escenic-admin** and how to use it. It describes how you can use the application to carry out various tasks, but does not in general discuss the purpose of the tasks: this is covered either in the later chapters of this manual or in the Escenic Content Engine Installation Guide.

When the Content Engine is running, you can access **escenic-admin** by starting a browser and pointing it at:

`http://`*your-server***`:8080/escenic-admin/`**

where *your-server* is the domain name or IP address of the server on which the Content Engine is running.

This should display the following page:



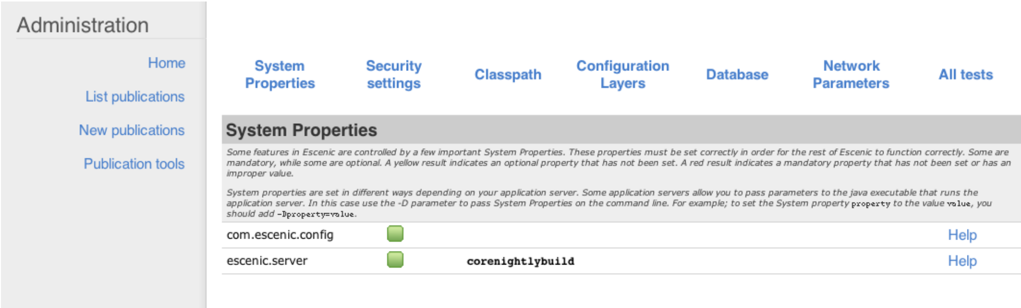
The menu on the left switches the display between four main pages, **Home**, **List publications**, **New publications** and **Publication tools**. These pages are described in the following sections.

2.1 Home

This page contains a long list of links that provide access to various system administration tools and services, described in the following sections.

2.1.1 Status

This option displays the Content Engine status page, which looks something like this:



This page displays the results of various sanity checks performed to determine the status of the Content Engine and is a useful diagnostics tool, particularly during the initial installation and configuration phase. The test results are grouped into seven different categories (**System Properties**, **Security settings**, **Classpath**, **Configuration Layers**, **Database** and **Network Parameters**), displayed in a menu across the top of the screen.

The result of each of the tests displayed on these pages is indicated by one of the following icons:

The test was passed, no action needed.

The test was not passed but the failure is not critical. Click on the **help** link for information about the consequences of the failure and how to fix the problem (if necessary).

The test was not passed and the failure is critical (that is, the Content Engine will not function properly until the problem is fixed). Click on the **help** link for information about the consequences of the failure and how to fix the problem.

For each test there is a **help** link on the right hand side of the window that displays information about the test: what the test does, what the consequences of failure are and advice on fixing failures.

2.1.2 Configuration Layer Report

This option displays a page that shows the settings of the Content Engine's mandatory configuration parameters. In the same way as the **Home** > **Status** page, it indicates whether each parameter is correctly set and provides **help** links with background information about each setting.

The Content Engine has many more configuration parameters than the ones shown here: to see the settings of other parameters, use the **Home** > **Component Browser** option (see section 2.1.15).

The Content Engine has a **layered** configuration system that allows more specific configuration parameter settings (for example, host-specific settings) to override more generic (for example, installation-wide) ones. It is therefore not always immediately obvious where a particular parameter setting originates from, or where the best place to modify it is. To be a successful Escenic server administrator you need to understand this configuration system. It is described in chapter 4.

2.1.3 View Installed Plugins

This option displays a list showing the status of all the plug-ins currently installed with the Content Engine. Here is an example of a plug-in status listing, in this case for the menu editor plug-in:

menuEditor	2.0-SNAPSHOT	The Escenic Content Engine MenuEditor	The Menu editor is a Browser based GUI used to edit "menu.xml" files in the publication.

Type	Target	Task	Area	Roles	Label	LabelKey	Uri
internal-link	escenic	/main-menu	plugins	[...]	Menu editor	null	/plugin/menuEditor/editMenu.do

The green check mark indicates that the plug-in is correctly installed. Badly installed plug-ins are marked with a icon instead.

2.1.4 Performance Summary

This option displays a page of Content Engine performance data.

At the top of the page are controls for determining how the page is refreshed:

Refresh
: This button refreshes the page **now**. It can be used at any time, but will normally only be used when the **Auto Refresh** option is disabled.

Auto Refresh
: Check this option if you want the page to be automatically refreshed every 2.5 seconds. If this option is not checked them the page is only refreshed on request.

The contents of the **Performance Summary** page is divided into separate sections for each of the applications running on the application server: one section (called **Global**) for the Content Engine itself, and one for each related application (including publication applications). The **Global** section contains **Cache**, **Load Averages** and **Activity Monitors** information. The other sections usually only contain **Cache** information.

2.1.4.1 Cache Summaries

A cache summary has the following columns:

Component Name
: The name of the component that manages the cache. The name is also a link to the component's component browser page, where you can tweak the cache settings. For information about the component browser, see section 2.1.15. For advice on cache tuning, see chapter 6. Note that any changes you make to cache settings using the component browser are temporary and will be lost the next time the Content Engine is restarted. To make permanent changes to a cache's settings you must edit a `.properties` file in one of your configuration layers (see chapter 4).

Size
: The maximum number of entries allowed in the cache.

Adds
: The number of entries added to the cache since the last restart.

Hits
: The number of hits (successful cache look-ups) since the last restart.

Misses
: The number of misses (unsuccessful cache look-ups) since the last restart.

Idle
The average time taken for an idle object to pass through the cache, in milliseconds.

Cache Health
A general indicator of how well the cache is performing. The vertical bar shows what proportion of the items in the cache are popular (popular items are ones which keep being requested and therefore stay in the cache for a long time). The green area in the center of the graph indicates the "healthy" area, and the vertical bar should mostly appear within this area. If the indicator is to the left of the green area, then almost all of the objects in the cache are popular. This suggests that the cache may be too small, and there are even more popular objects that cannot be kept in the cache because it keeps filling up. If the indicator is to the right of the green area, then very few of the objects in the cache are popular, suggesting that the cache is larger than it needs to be.

Note that you should not make changes to a cache's size based on a single reading of this indicator. You need to observe the indicator over time, and only make an adjustment if the indicator is consistently outside the healthy area.

LRU Distribution
This graph shows the distribution of items in the cache the last time the cache was full and needed emptying. Each bar represents a level of popularity, so the first bar indicates how many items were very popular (frequently requested), and the last bar shows how many objects were very unpopular. A well-functioning cache should have most items at the left hand (popular) end. If the distribution seems to be completely even it may mean that the cache is too small or too large. Consult **Cache Health** for further guidance, **Idle** to see whether or not the cache is retaining items for a sensible amount of time, and **Adds** to make sure that items are not moving through the cache too fast.

Popularity Distribution
This graph shows the relative popularity of the items in the cache the last time the cache was full and needed emptying. Popular (recently requested) items are shown at the left hand end, unpopular ones at the right hand end. A well-functioning cache should have most items at the left hand (popular) end.

Live hit rate
This shows the percentage hit rate of the cache since the last time the **Performance Summary** page was updated. In other words, if **Auto Refresh** is switched on, it shows the hit rate over the preceding 2.5 seconds. If **Auto Refresh** is switched off then when you click **Refresh**, it shows the hit rate since the previous time you clicked **Refresh**.

2.1.4.2 Load Averages

The load averages table shows information about the load on various parts of the Content Engine. The table contains the following columns:

Component Name
The name of the component that monitors this part of the Content Engine. The name is also a link to the component's component browser page, where you may possibly find more detailed information than is displayed in the load averages table.

Success
The number of successful requests handled by this part of the Content Engine since the last restart.

Failures
The number of failed requests handled by this part of the Content Engine since the last restart.

Time
The amount of time spent in this part of the Content Engine since the last restart.

Load average
A graph showing the average load exerted on this part of the Content Engine over the last minute or so (assuming **Auto Refresh** is switched on - otherwise the length of time represented by the graph will depend on how frequently you have clicked on the **Refresh** button).

Description
The part of the Content Engine monitored by this component.

2.1.4.3 Activity Monitors

The activity monitors table shows information about the throttles used to limit the load on various parts of the Content Engine. The table contains the following columns:

Component Name
The name of the component that controls this throttle. The name is also a link to the component's component browser page, where you can adjust the throttle settings if necessary. For information about the component browser, see section 2.1.15. For advice on throttle tuning, see chapter 8. Note that any changes you make to throttle settings using the component browser are temporary and will be lost the next time the Content Engine is restarted. To make permanent changes to a throttle's settings you must edit a `.properties` file in one of your configuration layers (see chapter 4).

Current usage
The number of requests currently being handled by this part of the Content Engine.

Limit
The maximum number of concurrent requests allowed by the this throttle.

Description
The part of the Content Engine controlled by this throttle.

2.1.5 System Properties

This option displays a list of system-wide property settings.

2.1.6 View Services

This option displays a status page showing the current status of various services that the Content Engine depends on. On the right hand side of the page are various check boxes that you can use to control the information displayed on the page.

2.1.7 Issue a Support Request

Whenever you send a support request to Escenic, you should include full information about your current server setup. The simplest way to do this is to:

1. Select this option.
2. Copy the information listed on the displayed page.
3. Paste the information into the body of a mail.
4. Send the mail to support@escenic.com.

In some browsers you can create the mail automatically by clicking on the **send all this as an e-mail** link on the displayed page.

2.1.8 Create a Thread Dump

Displays a page content a thread dump from the server. You may in some circumstances be asked to supply a thread dump in connection with a support request. Simply copy the contents of this page and send it in a mail to support.

2.1.9 Top

This option displays a constantly updated list of the most active JSP templates. The list shows the amount of time spent in each listed JSP file during the preceding two seconds. It can be a useful tool for identifying bottlenecks in your JSP code.

2.1.10 View the Browser Log

This option displays the messages generated by Escenic templates. The messages displayed can come from two possible sources:

- Escenic tag library tags
- Template code. Template developers can explicitly include log messages in their templates using the **`util:logMessage`** tag.

Log messages are classified into various error level categories (**`ERROR`**, **`WARNING`** and so on). You can select which of these levels are to be displayed here using the **View the logging levels** option (see section 2.1.11).

2.1.11 Configure Logging Levels

This option display the Escenic logging level editor, which you can use to control what kinds of messages are added to the browser log (see section 2.1.10). All messages have two properties that are used by the logging level editor:

`category`

This is a string that identifies the source of the message. If the source is a Java program (which is usually the case), the string is the fully qualified class name of the class that issued the message (**`com.escenic.presentation.servlet.BootstrapFilter`**, for example). Messages generated by template code, on the other hand, have category strings defined by the template developer: template developers are recommended to follow a similar "dotted" naming convention.

`level`

This is a keyword denoting the severity of the condition that caused the message to be issued. The severity levels (from highest to lowest) are:

`FATAL`

indicates that a fatal error has occurred.

`ERROR`

indicates that a non-fatal error has occurred.

`WARN`

indicates that a possibly undesirable event has occurred.

INFO
: indicates that a event of possible interest has occurred.

DEBUG
: indicates that an event of possible significance in a debugging situation has occurred.

TRACE
: indicates that a traceable event has occurred.

The logging level editor lets you use these two message properties to control what messages are appended to the browser log. Messages are selected by assigning levels to categories. All messages belonging to that category that have the assigned level **or higher** will then be appended to the log. Assigning the level `ERROR` to the category `com.escenic.presentation.servlet.BootstrapFilter`, for example, will cause any `com.escenic.presentation.servlet.BootstrapFilter` messages with the level `ERROR` or `FATAL` to be appended to the log.

Instead of assigning one of the above level settings to `com.escenic.presentation.servlet.BootstrapFilter`, you can instead set the level to `INHERIT`. It will then inherit whatever level is set for `com.escenic.presentation.servlet`; if `com.escenic.presentation.servlet` is also set to `INHERIT`, then it will inherit whatever is set for `com.escenic.presentation` and so on. This means it is possible to set a general level for all messages by setting the level of the special category `root`, and then just set any exceptions as required.

2.1.11.1 Changing Logging Level

To change the setting of one of the categories listed in the editor, simply select the required logging level from the pull-down list on the right and click on **Apply changes**.

To change the setting of a category that is **not** listed in the editor:

1. Check **Show inherited categories**.
2. Click on **Apply changes**. This will cause all currently registered categories to be displayed, including all those have their level set to `INHERIT`.
3. Locate the required category and select the required level.
4. Un-check **Show inherited categories**.
5. Click on **Apply changes**. You will see that the category is now listed in the editor, because it has an explicit setting.

2.1.11.2 Adding Categories

Any categories defined in template code will not appear in the logging level editor, even when **Show inherited categories** is checked, unless they are explicitly added. To add a new category:

1. Enter the name of the new category in the **Enter new category** field.
2. Click on **Apply changes**.

The new category will initially be listed with its level set to `INFO`.

If template developers use the same "dotted" naming convention for their messages as is used for Content Engine messages, then the same inheritance rules are applied by the error logging system.

2.1.11.3 Filtering The Category List

If you check **Show inherited categories**, then the list of categories can be very long. You can limit the list to show only the categories you are interested in using the **Filter Categories** field. You can, for example, display only `com` categories by entering `com` in the **Filter Categories** field and then clicking on **Apply changes**.

2.1.12 View JSP Statistics

This option displays JSP-related performance statistics, and is mostly likely to be used by template developers rather than by system administrators. Statistics are only displayed if statistics gathering (or **profiling**) has been enabled in publication templates. For information about statistics collection and interpretation, see JSP Profiling.

2.1.13 Remove Objects From Cache

This option allows you to clear specific objects from specific caches (which can be useful for locating cache-related problems. You are recommended to use this option rather than the **Clear all caches** option (see section 2.1.14) if possible, as **Clear all caches** can have a significant effect on performance.

To use this option:

1. Select the type of object to be removed from the caches.
2. Select the caches from which the selected objects are to be removed.
3. Either:
    - Enter an SQL query that will return the IDs of the objects to be removed, or
    - Enter the IDs of the objects to be removed.
4. Click **Preview**. The selected object IDs displayed for confirmation.
5. If you are satisfied with the displayed object IDs, click **Confirm**.

2.1.14 Clear All Caches

This option empties all the Content Engine's caches.

> This option can have a significant effect on performance. You are advised to avoid using it on live systems. If possible, use the **Remove some objects from cache** option instead (see section 2.1.13).

2.1.15 Component Browser

This option displays the Escenic **component browser**. The component browser is a web application that you can use to:

- View current configuration parameter settings of the Content Engine, its associated web applications and publications.
- Find out where the current configuration parameter settings come from (that is, which particular configuration files they are set in).
- Temporarily modify configuration parameter settings.

Content Engine components are uniquely identified by fully qualified names consisting of a path and a name. The Content Engine's article list cache component, for example, has the following fully qualified name: `/neo/io/content/cache/ArticleListCache`. The components, in other words, are effectively organized in a tree structure. The component browser lets you navigate this tree structure and view the properties of Content Engine components.

To view the properties of the `ArticleListCache` component, for example, you would need to click on **Component browser** > **neo** > **io** > **content** > **cache** > **ArticleListCache**. A page of information about the `ArticleListCache` component is then displayed. It is divided into three sections: **Properties**, **Methods** and **Service Information**.

2.1.15.1 Properties

The properties section of a component browser page lists the current property settings of a component.

To change the setting of a displayed property:

1. Click on the property name. A new page is displayed, possibly containing a **New Value** field.
2. Enter a new value in the **New Value** field (if displayed).
3. Click on **Submit Query**.

- Not all properties are editable. If a property cannot be edited, then no **New Value** field is displayed when you click on the property name.
- Changes you make in this way are temporary and will be reverted the next time the server is restarted.
- Be careful! Don't change property settings on a live system unless you are sure you know what you are doing.

2.1.15.2 Methods

The methods sections of a component browser page lists the component's methods.

To execute a displayed method:

1. Click on the method name. A new page is displayed that contains a button or **Invoke** link for invoking the method, and may also contain fields in which you can enter method parameters.
2. Enter any required parameter values.
3. Click on the invocation button or link.

- Changes you make in this way are temporary and will be reverted the next time the server is restarted.
- Be careful! Don't execute methods on a live system unless you are sure you know what you are doing.

2.1.15.3 Service Information

Component properties are set during system start-up: the Content Engine reads them from `.properties` files. These files are named in the same way as the components they configure. The properties of the `/neo/io/content/cache/ArticleListCache` component for example, are loaded from files called *configuration-root*`/neo/io/content/cache/ArticleListCache.properties` that contain appropriate property settings such as:

```
maxSize=300
```

The Content Engine has a **layered** configuration system in which such property settings are loaded from a number of different locations. During start-up, the Content Engine searches through a series of locations (or *configuration-roots*) in turn and applies the settings it finds. The final property settings displayed in the component browser, therefore, are a result of merging all the settings found in these various locations. If a particular property is set in several locations, the last setting wins.

The service information section contains listings of all the **`.properties`** files loaded for a component, in the order they were loaded. You can therefore use this section to find out where particular properties are actually set.

For more information about the Content Engine's configuration system, see chapter 4.

2.1.15.4 Browsing Application and Publication Components

By default, the component browser displays the Content Engine's component hierarchy. You can, however, also use it to examine the component hierarchy of any web applications supplied with the Content Engine (the indexer web application, for example) or the component hierarchy of any publication.

When you are browsing the Content Engine's own component hierarchy, **Scope: Global** is displayed at the top of the component browser page. To display a different component hierarchy, click on the **Browse other scope** link displayed below this heading, then select the name of the "scope" (i.e., application or publication) you want to browse.

2.1.16 Database Browser

This option displays the Escenic **database browser**. The database browser provides a simply interface for submitting SQL queries to the database.

To use the database browser:

1. Enter an SQL query in the **Enter SQL Query** field.
2. Click on **Submit Query**.

The results of the query are then displayed on the page. The **Enter SQL Query** field is displayed below the results (it may be off-screen), so you can enter another query. All valid queries you enter are listed **below** the **Enter SQL Query** field: you can re-use them by clicking on them or remove them from the list by checking the **Remove** check box before you click on **Submit Query**.

Click on **Clear** to clear the **Enter SQL Query** field. Click on **Reset** to recall the last valid executed query.

> This interface is provided to facilitate browsing of the Escenic database, not editing. Do **not** execute any query that modifies the contents of the Escenic database.

2.2 List publications

This page lists all the publications currently served by the Content Engine, and provides various publication management tools.

It contains the following links:

Select all
: Selects all listed publications.

Deselect all
: Deselects all listed publications.

Invert checkbox selection
: Selects all currently unselected publications and deselects all currently selected publications.

Information
: Displays page containing useful information about one of the listed publications, and links for accessing it in various ways.

Run field indexer
: Generates the indexes used by the `article:list` tag. For information about why and when you would want to use this option, see Generating Content Item Field Indexes.

Update resources
: Updates the resources of all currently selected publications. For further information about this process, see section 2.2.1.

Delete
: Deletes all currently selected publications. A new page is displayed on which the names of all the selected publications are listed. To complete the operation, click **Confirm**.

2.2.1 Update Resources

The structure and characteristics of an Escenic publication are defined in a set of files that are collectively referred to as **publication resources**. When a publication is created, a set of publication resources must be uploaded to the Content Engine as a basis for the publication. Changes to an existing publication may often require these publication resources to be modified. The **Update resources** option allows publication resources to be modified by overwriting them with new versions.

For detailed information about the various publication resources, see the Escenic Content Engine Resource Reference.

The usual procedure for updating publication resources is:

1. Prepare the updated resources and place them in a known location on your local machine ready for upload.
2. On the `escenic-admin` **List publications** page, select all the publications that are to be updated (you may have several publications based on the same resource set).
3. Select **Update resources**. A page containing the message "You have to **upload** a resources first!" is usually displayed.
4. Click on **upload**. The **Upload resources** page is then displayed (see section 2.5).
5. Select the correct resource type for the resource you intend to upload.
6. Click on **Browse...** and locate the resource you intend to upload.
7. Click on **Upload**.
8. If the resource is successfully uploaded and validated, click on **List publications**.
9. Repeat steps 2 and 3. This time, since you have now uploaded a resource, the "You have to **upload** a resources first!" message is not displayed. Instead, the resource(s) you have uploaded

and the publications you have selected for update are listed. To update the listed publications, click **Confirm**.

2.3 New publications

This page displays forms for creating new publications and for uploading the resources required to create them.

The structure and characteristics of an Escenic publication are defined in a set of files that are collectively referred to as **publication resources**. When a publication is created, a set of publication resources must be uploaded to the Content Engine as a basis for the publication. For detailed information about the various publication resources, see the Escenic Content Engine Resource Reference.

The usual procedure for creating a new publication is:

1. Prepare a publication WAR file containing the required resources and place it in a known location on your local machine ready for upload.
2. In `escenic-admin`, select **New publications**. The **Upload resources** page is then displayed (see section 2.5).
3. Select **Publication Definition** resource type option.
4. Click on **Browse...** and locate the publication WAR file.
5. Click on **Upload**.
6. If the WAR file is successfully uploaded and the resources in it are successfully validated, click on **create a publication**. This displays the **Create Publication** form.
7. Enter a name for the publication in the **Publication Name** field.
8. Select the required **Publication type**:

    default
    For an ordinary publication.

    configuration
    For a Widget Framework template set.
9. Do one of the following:
    - Either enter a password for the publication administrator in the **Administrator password** field and enter it again in the **Verify password** field.
    - Or check **Create publication without password**.

    > You should only check **Create publication without password** if you have set up third party authentication (see chapter 15) for Web Studio. Checking this option does **not** mean users will be able to log in to Web Studio without entering a password. It just means that no password is stored in the Content Engine database, so if you don't set up third party authentication and create a user in the third party authentication system with the administrator user name, then nobody will be able to log in as administrator of the publication.
10. Click on **Submit**.

It is also possible to upload the resources needed to create a publication individually, rather than uploading them all together in a WAR file. For information about this and a more detailed description of the **Upload Resources** page, see section 2.5.

> This section describes how to create a single publication. In order to be able to use the publication, you must also deploy the web application that will drive the publication (and possibly many other similar publications). For information on how to deploy publication web applications, see Deployment.

2.4 Publication tools

This page contains a list of links that provide access to publication administration tools, described in the following sections.

2.4.1 Manage Tag Structures

This option displays a tag management page. You can use this form to create and import **tag structures**.

A tag structure is a hierarchical structure of **tags**, Tags are keywords that can be used to classify publication content for search and retrieval purposes. You might, for example tag a travel article about Thailand with the tags **`Travel`** and **`Thailand`**. Tags are organized as hierarchies in order to be able to represent logical associations between the concepts they represent. If the tag **`Thailand`**, for example is a child of another tag called **`Asia`**, then a search for content using the tags **`Travel`** and **`Asia`** would return our article (possibly along with other travel articles about other Asian countries).

You can create as many tag structures as you wish, organized in any way you wish. You might for example create separate tag structures for places, sports, genres, politics and so on. Once you have created a tag structure, you can add tags to it in two ways:

- Import tags defined in XML files (see section 2.4.1.2)
- Allow Content Studio users to add tags to a structure on an ad-hoc basis (see Tag Structures)

2.4.1.1 Create a Tag Structure

If you have not previously created any tag structures, then the tag management page contains a single form called **Create new Tag Structure**. To create a tag structure fill in the form's fields as follows and click on **Create**:

`Scheme`

A tag structure is uniquely identified by its **scheme**, a specially formatted identifier string that must conform to the entity portion of RFC 4151. You can create a valid scheme by conforming to the following format:

```
structure-name.domain-name,yyyy
```

where:

structure-name

is a name for the structure. The name must not contain any spaces or special characters other than '-' and '.' (the same rules apply as for domain names).

domain-name
is a domain name that is or has been owned by your organization.

yyyy
is one of the years in which *domain-name* was owned by your organization.

You might, for example, create tag structures with the following scheme names:

```
places.mycompany.com,2011
sports.mycompany.com,2011
genres.mycompany.com,2011
```

`Name`
The tag structure's display name. This is the name that users will normally see, for example: **`Places`**, **`Sports`** or **`Genres`**.

`Description`
A description of the tag structure and its purpose. The description for **`genres.mycompany.com,2011`**, for example, might be:

```
Book, film and theatre (but not music) genres
```

As long as the scheme you specified is syntactically correct and does not already exist, the new scheme is created, and appears in a **Tagging Structures** section (see section 2.4.1.2) below the **Create new Tag Structure** form.

> You can (in theory) also use this form to create tag structures based on tag collections maintained by other organizations. In this case you would enter the other organization's scheme name in the scheme field. You could then convert their tags to the tag syndication format described in section 2.4.1.3 and import it. Since this tag structure is maintained by a different organization, you would then need to ensure that your users do not modify the imported structure. In practice, there are currently very few standard tag collections available, so this possibility is not likely to be of much interest.

2.4.1.2 Tagging Structures

All existing tag structures are listed in a **Tagging Structures** section below the **Create new Tag Structure** form.

Two buttons are available for each tag structure:

Import
Click on this button to import tags from a tag syndication file to this tag structure. A new form is displayed that you can use to locate the file on your local machine and upload it to the Content Engine. See section 2.4.1.3 for a description of the tag syndication file format.

> You can only use this function to add tags to a structure, not to update existing tags. Any tags in an imported file that have the same **`term`** (that is, id) as an existing tag in the structure are ignored.

Delete
Click on this button to delete this tag structure. When you delete a tag structure:

- All its member tags are deleted
- The deleted tags are removed from all content items in which they have been used.

When you click on this button, a confirmation form is displayed that indicates how many tags the structure contains, and how many content items will be affected by the deletion. To complete the deletion, click on **Yes**.

2.4.1.3 classification-tags

The `classification-tags` schema defines the **Escenic tag syndication format**. The Escenic tag syndication format can be used to import hierarchically structured tags into a predefined Escenic tag structure.

Namespace URI

The namespace URI of the `classification-tags` schema is `http://xmlns.escenic.com/2011/classification-tags`.

Root Element

The root of a `classification-tags` file must be a `tags` element.

2.4.1.3.1 alias

An alias of the tag `tag` element.

Syntax

```
<alias>
  text
</alias>
```

2.4.1.3.2 description

A description of the meaning and purpose of the tag represented by this element's parent `tag` element.

Syntax

```
<description>
  text
</description>
```

2.4.1.3.3 label

The label of the tag represented by this element's parent `tag` element. A tag's label is the string displayed in user interfaces. There are no restrictions on the characters used in a label: spaces, punctuation marks and special characters are all allowed.

Syntax

```
<label>
  text
</label>
```

2.4.1.3.4 tag

Represents a tag.

Syntax

```
<tag
    term="text"
    parent-term="text"?
  >
  <label>...</label>

  <description>...</description>?
  <alias>...</alias>*?
</tag>
```

Examples

- This example shows a root-level **tag** element with a **description**.

```
<tag term="eu">
  <label>European Union</label>
  <description>The European Union</description>
</tag>
```

- This example shows a **tag** element with a **parent-term** attribute, but no **description**.

```
<tag term="england" parent-term="uk">
  <label>England</label>
</tag>
```

Attributes

term="*text*"
: A locally unique identifier for the tag represented by this **tag** element. "Locally unique" means in this case that the tag must be unique not only within this tag syndication file, but also within the tag structure to which it is being imported (the target structure may already contain a number of tags). The term may not contain any spaces or any special characters other than ".", "-" and "_".

parent-term="*text*" (optional)
: A reference to the **term** of another tag under which the tag represented by this **tag** element should be inserted. If this attribute is not specified then this tag will be created as a root-level tag.

2.4.1.3.5 tags

The root element of an Escenic tag syndication format file.

Syntax

```
<tags>
  <tag>...</tag>+
</tags>
```

Examples

- This example shows a **tags** element containing a small number of **tag** elements.

```
<tags>
  <tag term="eu">
    <label>European Union</label>
    <description>The European Union</description>
```

```
  </tag>
  <tag term="uk" parent-term="europe">
    <label>United Kingdom</label>
    <description>The United Kingdom of Great Britain and Northern Ireland</
description>
  </tag>
  <tag term="england" parent-term="uk">
    <label>England</label>
  </tag>
  <tag term="fr" parent-term="eu">
    <label>France</label>
  </tag>
  <tag term="tag2">
    <label>Norway</label>
    <alias>Norge</alias>
  </tag>
</tags>
```

2.4.2 Grant a User Read/Write Permission

This option displays the **Grant a user read/write permission** form. You can use this form to:

- Change the access rights of an existing user
- Create a new user and grant it access to publications

To change the access rights of an existing user:

1. Select **Existing user**.
2. Enter a user name in the the **User name** field.
3. Click **Next** to display the second form.

To create a new user:

1. Select **New user**.
2. Enter the new user's credentials in the appropriate fields.
3. Select the publication to which the new user is to belong.
4. Click **Next** to display the second form.

The second form contains a list of all publications at the site. Select/unselect access rights as required and click on **Save**.

If you need to assign other roles than **Reader** or **Editor**, then you must use Web Studio to do it. See [Global Roles](#) for details.

2.4.3 Export Publication Content

This option displays the **Export from publication** form. You can use this form to export an entire publication or selected parts of a publication to Escenic syndication format files. To export content from a publication, enter your requirements in the form and click on **Export**.

Use the controls in the form as follows:

Publication ID
: Enter the ID of the publication from which you want to export content.

Section IDs
: Enter a comma-separated list of the sections from which you want to export content. If you leave this field empty, then content will be exported from all sections.

Folder name
: The path of the folder to which the exported files will be written. You can specify either an absolute or a relative path. Relative paths are relative to the `java.io.tmpdir` system property.

Group files by object type
: Check this option if you want different object types (e.g., content items, sections, section pages) to be written to separate output files. Section pages, inboxes and lists are all based on the same internal object type, and will therefore be written to the same file.

Maximum items per file
: If you don't want to generate very large syndication files, you can limit the size by specifying the maximum number of content items/sections etc. to be written to a file. If this limit is reached, then several files will be generated.

Export sections
: Check this option if you want sections to be exported.

Export content items
: Check this option if you want content items to be exported. If you only want certain types of content item to be exported, enter a comma-separated list of content type names in the **Content types** field. If you leave this field empty then all content types will be exported.

Export pools
: Check this option if you want section pages, lists and inboxes to be exported.

Export from time/Export to time
: You can use these fields to limit the export to objects that have been modified within a specific period of time. You can, for example, only export those objects that have changed or been added since the last export was carried out.

2.4.4 Resolve Unresolved Relations

This option allows you to resolve **unresolved relations**. An unresolved relation is a content item that has a "dangling" relation to another content item: the other content item has not yet been located, so the relation is incomplete. Unresolved relations should not normally occur, but can arise after import operations. To resolve all unresolved relations, simply click on **Confirm**. Any relations that cannot be resolved (because the referenced content item cannot be found) are left unchanged.

2.5 Upload Resources

This page is displayed both when updating publication resources using the **Update resources** option (see section 2.2.1) and when creating new publications using the **New publications** option (see section 2.3).

To upload resources using this page you must:

1. Specify the type of resource you are going to upload by selecting one of the **Type of resource** options

2. Either enter the path of the resource to be uploaded in the **File to upload** field or else click on the **Browse...** button and locate the resource using the displayed file browser dialog.
3. Click on **Upload**.

The resource type options are:

Publication definition
: A publication WAR file is to be uploaded. A publication WAR file contains all the resources needed to define an Escenic publication. It will also usually contain the JSP templates defining the web application that drives the publication, and may contain syndication files with content to be imported into the publication. This is the option you usually choose when creating a new publication (although you can also use it when updating existing publications). It is a convenient means of importing all the resources in one go. The JSP templates, which are not required for the purpose of creating new publication or updating resources are simply ignored.

Content type definitions
: A content-type resource is to be uploaded. This is an XML file defining all the content types a publication may contain.

Feature definitions
: A feature resource is to be uploaded. This is a plain text file containing property settings that set various Content Engine features for a publication.

Image version definitions
: An image-versions resource is to be uploaded. This is an XML file defining all the different versions of images that a publication may contain.

Layout definitions
: Not in use.

Layout group definitions
: A layout-group resource is to be uploaded. This is an XML file defining the layouts to be used on a publication section pages.

Content definitions
: A syndication file is to be uploaded, containing content to be imported to the publication. For general information about syndication files, see the Escenic Content Engine Syndication Reference.

Other type of resource
: Select this option if you want to upload any other resource types (for example, a plug-in resource type). You must then enter a string identifying the resource type in the **Please specify** field.

The **Upload** option not only uploads the specified resources, it also validates them. After the upload operation, the page is redisplayed, this time with an **Available Resources** section that contains a list of currently uploaded resources showing their validation status. Any resource that fails to validate is marked **Not valid**, and followed by an error message providing some indication of what the problem is. If this happens, correct the error and upload a new version of the resource.

If you want to upload several resources you can either package them in a publication WAR file and upload that or else select the **Upload** option several times to upload them individually.

If you upload a complete set of publication resources that is sufficient to create a publication, then a **Create Publication** section appears on the page, containing the message "You now have enough resources to **create a publication**". To create a publication from these resources, click on the **create a publication** link.

3 The indexer-webapp Web Application

An indexing web application called **`indexer-webapp`** is included with the Escenic Content Engine. It receives content items passed to it by one of the Content Engine's indexer web services and passes them on to Solr, the search engine used by Content Studio (and also by most publication web applications). This chapter contains a brief description of the **`indexer-webapp`** administration interface and how to use it.

When the **`indexer-webapp`** is running, you can access its administration interface it by starting a browser and pointing it at:

`http://`*`your-server`***`:8080/indexer-webapp/admin/`**

where *your-server* is the domain name or IP address of the server on which the **`indexer-webapp`** is running.

The administration interface is a single page divided into the following sections:

- Configuration
- Current state
- Current Statistics
- Indexer actions

displays information about the configuration and current status of the indexer, plus four buttons you can press to affect the operation of the indexer.

For more information about the current state of the search engine, visit the Solr administration page by pointing your browser at:

`http://`*`your-server`***`:8080/solr/admin/`**

where *your-server* is the domain name or IP address of the server on which Solr is running. For information about about how to use this interface and general information about Solr, visit http://lucene.apache.org/solr/.

3.1 Configuration

This section displays the following information about the indexer's configuration:

Base Query URI
The URI of the Content Engine web service from which the documents to be indexed are read. This URI is set in the Tomcat configuration file **`context.xml`** (see Install Application Server).

Style sheet
The XSL stylesheet used to prepare documents for indexing.

Update URI
The URI of the Solr instance to which index updates are sent. This URI is set in the Tomcat configuration file **`context.xml`** (see Install Application Server).

3.2 Current State

This section displays information about the current state of the indexing process. If **Number of documents read but not yet processed** is 0, then indexing is complete. Click on your browser's **Refresh** button to update the displayed information.

3.3 Current Statistics

This section displays statistics about the indexing process.

3.4 Indexer Actions

Under normal operation, the indexer starts by indexing the most-recently modified content item and works backward to the least-recently modified content item. While it is doing so, new changes may be made: existing content items may be modified, new content items created. The indexer prioritizes the indexing of these newly-modified and newly-created content items, and interrupts the indexing of older content in order to deal with them. Eventually, however, the indexer will index the least-recently modified content item, and then only need to deal with incoming changes.

The buttons in the administration interface affect the indexing process as follows:

Reindex...
: Aborts the current indexing process (whether or not the indexer has succeeded in reaching the least-recently modified content item) and restarts it from the most recently modified content item. As it works backwards it will update the indexes of previously indexed content items.

  Re-indexing may be necessary for a variety of reasons (it is often required after installing a new version of the Content Engine).

  > Re-indexing may take a long time (possibly hours). During this period, searches executed in Content Studio may return incomplete results. In some production environments this may be unacceptable: see section 5.3 for a description of how to avoid the problem.

  Clicking on this button displays a new page containing the message **`Reindexing`**`....` To redisplay the administration page, simply click on your browser's **Back** button.

Pause Indexer
: Temporarily suspends the current indexing process. You can resume the process by clicking on the **Resume Indexer** button.

  Clicking on this button displays a new page containing the message **`Indexer is now paused`**`....` To redisplay the administration page, simply click on your browser's **Back** button.

Resume Indexer
: Resumes an indexing process that has previously been suspended using the **Pause Indexer** button.

Optimize Solr Index
: Optimizes the index. Old indexes can become fragmented and disorganized. Selecting this option sends an optimization request to Solr. Solr then creates a new, reorganized and optimized copy of the existing index. When the optimized copy is complete, the old index is deleted.

> Do not select this option unless you are certain that there is sufficient disk space available on the Solr host. (In order to optimize an index you need enough free disk space to hold another two copies of the index.)

Clicking on this button displays a new page containing the message `Optimizing index...`. To redisplay the administration page, simply click on your browser's **Back** button.

4 Configuring The Content Engine

For configuration purposes, the Content Engine is regarded as a hierarchy of configuration objects representing various parts of of the system. These configuration objects are called called **components**. Each component has properties that can be set in a corresponding configuration file. The configuration files are standard Java properties files with a well-defined format (see the Javadoc description of java.util.Properties.load(java.io.InputStream)).

The configuration files are stored in a folder tree that reflects the component hierarchy. At the top of a Content Engine configuration tree, for example, you will find files such as **`ServerConfig.properties`**, containing very general configuration settings. At the bottom of the folder tree are files such as **`/etc/escenic/engine/common/com/escenic/webservice/search/DelegatingSearchEngine.properties`** that contain detailed settings for very specific parts of the system.

4.1 Configuration Layers

The Content Engine's configuration system is not only hierarchical, it is also **layered**. What this means is that a Content Engine installation can contain several configuration trees in different locations. These trees can be considered as layers because they are read in sequence, each layer adding new property settings or overwriting settings already made in lower layers. Right at the start of the configuration process, the Content Engine loads a special configuration layer called the **bootstrap layer**, which configures the configuration process itself. It does this by defining:

- How many configuration layers there are
- The relative priority of the layers
- Where the layers are located

Once this has been done, the various layers are loaded in turn and merged into the final server configuration.

The purpose of this layering is to simplify both the upgrade process and the management of large multi-server installations as follows:

- The Content Engine is delivered with a **default configuration layer**, which has lowest priority, and an **add-on configuration layer** that can be used by add-ons to make any changes that they require. You should never modify these layers, since they are overwritten when the Content Engine and/or add-ons are upgraded, and your changes will be lost.
- Also delivered with the system is a **skeleton configuration layer** that you can use as a basis for creating configuration layers of your own. You will need to create at least one site-wide configuration layer called the **common configuration layer**. In this layer you can override default settings that do not meet your site's requirements.
- If you are running a multi-host site, you will also probably need to create additional configuration layers for each host that override any properties for which host-specific settings are required. These are referred to as **host configuration layers**.
- You can create even more layers: on large multi-host sites you may have "families" of hosts that perform the same function, and therefore have many configuration settings in common. It may

then make sense to create **family configuration layers** between the common configuration layer and the host configuration layers.

Note that the individual layers do not need to be complete: a layer can consist of just one `.properties` file, and a `.properties` file does not need to contain settings for all of a component's properties.

Configuration layers can be loaded from three different types of location or **depot**:

- JAR files in the classpath
- Explicitly specified JAR files
- Specified file system locations

The default configuration layer and the plug-in configuration layer are loaded from JAR files in the classpath.

You are recommended to create your common configuration layer (and any other layers you need) in the file system, ideally in the `/etc/escenic/engine` folder. The delivered bootstrap layer is configured to look for your configuration layers in this location. For detailed information on how to create configuration layers, and how to modify the bootstrap layer so that they are read in the correct order, see section 4.3.

4.2 Configuration File Format

A configuration file consists of a sequence of assignments of the form:

```
keyword=value
```

Each assignment must appear on a separate line. Lines can however be broken by using the backslash (\) as a continuation character. The use of the equals sign is optional (it can be replaced by white space). Otherwise white space is ignored.

Lines that start with either "#" or "!" are treated as comments and discarded.

In most cases:

keyword
: is the name of a property

value
: is the value to be assigned to the property

One of the *keywords* may be the special keyword `$class`. In this case *value* must be the fully qualified name of a class. This tells the system to create an object of the specified class: the properties specified in the rest of the file are assigned to this object. A complete property file **must** in fact include such an assignment, since there must be an object to assign properties to. However, this assignment is always included in the default layer configuration files, so it can usually be omitted from configuration files in higher layers. (Note, however, that if you add a configuration file to one of your layers that does not exist in any of the supplied lower layers, then this class assignment is required.)

Complex Properties

Most properties in the configuration files have simple values such as integers, string or booleans. More complex assignments can be made, however:

Component objects

Components can be "wired together" by means of property assignments. Component A, for example, may have a property that needs to be set to reference component B. This kind of property can be set by an assignment of the following form:

```
keyword = component-path/component-name
```

For example:

```
otherComponent = /mycomponents/Important
```

The component path does not have to be absolute. You can also specify a path relative to the folder of the current component. For example:

```
otherComponent = ../../Important
```

Arrays

Array properties can be set by separating the values in the array with commas, for example:

```
numbersToCheck = 10,20,30,45,70
```

Maps

Mapped properties can be set by a series of assignments of the following form:

```
keyword.key = value
```

For example:

```
component.3 = /mycomponents/Important
component.2 = /mycomponents/LessImportant
component.1 = /mycomponents/Unimportant
```

Note that mapped properties are set in alphabetical key order (1, 2, 3 in this case), not the order in which they appear in property files. This ensures a fixed order of creation even when the assignments are spread across several configuration layers.

Variables

The values assigned to properties can include placeholders for variables. When the property is assigned, the placeholder is replaced by the value of the variable it references. The syntax for a variable placeholder is:

```
${variable-reference}
```

Four different kinds of variable reference are supported:

System property references

variable-reference can be the name of any system property. For example:

```
myUrl = http://${escenic.server}:8080/my/page/
```

Component property references

variable-reference can be a reference to any component property. It must have the form:

```
component-path/component-name.property-name
```

For example:


```
myImportantValue=${/mycomponents/Important.value}
```

JNDI references

variable-reference can be a reference to any JNDI name. It must have the form:

```
jndi:jndi-name
```

For example:

```
providerUrl=${jndi:java:comp/env/PROVIDER_URL}
```

JNDI references are particularly useful as a means of creating configurations that can be used in more than one environment (both a test environment and production environment)

Environment variable references

variable-reference can be a reference to any operating system environment variable. It must have the form:

```
env:environment-variable
```

For example:

```
importantOsValue=${env:IMPORTANT}
```

Configuration File Encoding

Configuration files, in accordance with the rules for standard Java properties files, must be encoded using the ISO-8859-1 character set. If you need to include characters outside this character set, then you can do so using the following syntax:

```
\uxxxx
```

where *xxxx* is the hexadecimal Unicode value of the required character. The use of the backslash as an escape character to introduce Unicode values and as a continuation character means that you must always repeat any backslashes that you want to appear in the file. This Windows path, for example:

```
C:\my\windows\path
```

will be read as:

```
C:mywindowspath
```

unless you repeat the backslashes as follows:

```
C:\\my\\windows\\path
```

Example

The following example illustrates some the property types discussed above.

```
$class = com.mycompany.SomeClass
numbersToCheck = 10,20,30,45,70,\
                 131,199,343,546
otherComponents = ./Other
somePath = ${/ServerConfig.filePublicationRoot}/myroot
# Fruits
  fruit.apple     /mycomponents/Apple
  fruit.orange    /mycomponents/Orange
  fruit.banana    /mycomponents/Banana
```

It contains the following items:

- The creation of a component object.
- An array of numbers, with a line break.
- A reference to another component called **Other** in the same folder as this one.
- A path composed of the value of the **ServerConfig** component's **filePublicationRoot** property and the string value **/myroot**.
- A comment.
- A mapped property called 'fruit' with three values. Note that the properties will be created in alphabetical order, not the order which they appear. Also note the omission of the '=' sign, which is not required.

4.3 Managing The Configuration Layers

The first time the Content Engine is installed, the assembly tool's **initialize** target creates the bootstrap layer in **/opt/escenic/assemblytool/conf**. The bootstrap layer is predefined to look for the following configuration layers, and read them in the specified order:

1. **default layer** (in the delivered Content Engine JAR files)
2. **add-on layer** (in add-on JAR files)
3. **common layer** (in **/etc/escenic/engine/common**)
4. **family layer** (in **/etc/escenic/engine/family/***family-name*)
5. **host layer** (in **/etc/escenic/engine/host/***host-name*)

The following sections tell you how to:

- Create the common configuration layer
- Add a host configuration layer
- Add a family configuration layer
- Add further layers
- Change the location of a layer

4.3.1 Create The Common Configuration Layer

A skeleton configuration layer is provided in **/opt/escenic/engine/siteconfig/config-skeleton**. To create a common configuration layer from this skeleton, log in as **escenic** and copy the configuration layer to **/etc/escenic/engine/common**.

```
$ cp -r /opt/escenic/engine/siteconfig/config-skeleton/* /etc/escenic/engine/common/
```

You can now configure your whole Escenic installation by modifying the **.properties** files in the **/etc/escenic/engine/common/** tree.

4.3.2 Add A Host Configuration Layer

If your Escenic installation is spread across more than one host machine, then you will almost certainly need to set some properties differently on the different hosts. You can do this by creating a host configuration layer which is read after the common configuration layer. Any settings made in this layer will therefore override settings made in lower layers.

Obviously the contents of this layer need to be different for each host. The recommended method of doing this is to keep all your configuration layers (in fact the whole **`/etc/escenic`** tree) in a shared folder. If you have set up your system in this way, then you can create a set of host layers as follows:

1. Create an **`/etc/escenic/engine/host/`***host-name* folder for each host:

   ```
   $ mkdir -p /etc/escenic/engine/host/host-name
   ```

2. Copy the files containing the properties you are interested in overriding from the skeleton configuration layer to the corresponding relative location in each *host-name* folder.
3. Modify each of the copied **`.properties`** files as required.

This will work because the location of the host configuration layer is defined as follows in **`/opt/escenic/assemblytool/conf/layers/host/Files.properties`**:

```
fileSystemRoot=/etc/escenic/engine/host/${hostname env:HOSTNAME env:COMPUTERNAME
 "localhost"}/
```

If you are using a different location for your configuration layers, then you will need to modify this setting and redeploy (see section 4.3.5).

4.3.3 Add A Family Configuration Layer

In really large installations with many servers, you may decide that it makes sense to define "families" of hosts that have similar functions (a "publishing" family and a "presentation" family, for example), and define corresponding configuration trees that enable them to be controlled as a group. Any properties that you want to be the same for all publishing hosts can then be set once in this layer rather than being set separately for each host in the host configuration layer.

You can create a family configuration layer in the same way as a host layer:

1. Create an **`/etc/escenic/engine/family/`***family-name* folder for each family:

   ```
   $ mkdir -p /etc/escenic/engine/family/family-name
   ```

2. Copy the files containing the properties you are interested in overriding from the skeleton configuration layer to the corresponding relative location in each *family-name* folder.
3. Modify each of the copied **`.properties`** files as required.

The location of the family configuration layer is defined as follows in **`/opt/escenic/assemblytool/conf/layers/family/Files.properties`**:

```
/etc/escenic/engine/family/${com.escenic.config.engine.family "default"}
```

In order for this setting to work, the system property **`com.escenic.config.engine.family`** must be set on each host to the name of the family to which the host belongs.

If you are using a different location for your configuration layers, then you will need to modify this setting and redeploy (see section 4.3.5).

4.3.4 Add Further Layers

If you want, you can add further layers to create an even more flexible configuration system. To add an extra configuration layer between the family layer and the host layer, for example, you would need to:

1. Open **/opt/escenic/assemblytool/conf/Nursery.properties** in a text editor.
2. Change this setting:

   ```
   layer.05=/layers/host/Layer
   ```

   to:

   ```
   layer.06=/layers/host/Layer
   ```

3. Add a property defining your new layer (we'll call it "group") as layer 05:

   ```
   layer.05=/layers/group/Layer
   ```

4. Create two new **.properties** files: **/opt/escenic/assemblytool/conf/group/Layer.properties** and **/opt/escenic/assemblytool/conf/group/File.properties**. **/opt/escenic/assemblytool/conf/group/Layer.properties** should contain the following:

   ```
   $class=neo.nursery.PropertyFileConfigurator
   depot=./Files
   ```

   and **/opt/escenic/assemblytool/conf/group/File.properties** should contain:

   ```
   $class=neo.nursery.FileSystemDepot
   fileSystemRoot = /etc/escenic/engine/group/${escenic.group}
   ```

5. You can now create group configuration layers in exactly the same way as you created host and family layers, and use system properties to select the required layer in the same way.
6. Run the assembly tool.
7. Deploy the results.
8. Restart.

The bootstrap layer will never be overwritten by the assembly tool once it has been created, so any changes you make are persistent. If the bootstrap layer should ever be deleted, however, a new one can be created by running the assembly tool's **initialize** target.

> Do not insert your own layers below layer 03.

4.3.5 Change The Location of a Layer

To change the location of one of the layers:

1. Open the **File.properties** file for the layer you want to move. For example, to move the common layer, open **/opt/escenic/assemblytool/conf/common/File.properties**.
2. Edit the **fileSystemRoot** property to point to the required location.
3. Copy your common configuration layer to the new location.
4. Run the assembly tool.
5. Deploy the results.
6. Restart.

The bootstrap layer will never be overwritten by the assembly tool once it has been created, so any changes you make are persistent. If the bootstrap layer should ever be deleted, however, a new one can be created by running the assembly tool's **`initialize`** target.

5 Search Engine Configuration and Management

The Content Engine's search functionality is provided by Apache Solr, a Java-based open source search engine that runs as a web application alongside the Content Engine. A copy of Solr is bundled with the Content Engine, and if you follow the standard installation procedure described in the Escenic Content Engine Installation Guide, then a **`solr`** instance is deployed alongside every Content Engine you deploy. All Content Studio search functions depend on Solr, and Solr can also be used to drive the search functions in your publication web applications.

The use of an external search engine that is completely decoupled from the Content Engine ensures a high degree of flexibility. It is possible to configure the search engine and the other components involved in providing search functions in many different ways to meet differing requirements. The components involved in providing the Content Engine's search functions are:

indexer web services

Two indexer web services are provided by the Content Engine for logging changes to content managed by the Content Engine. The indexer web services are called:

`index`

This web service helps to maintain the internal index used by Content Studio and other editorial systems. Every time any content item is added, modified or deleted, it adds an entry to its change log. The entry contains the URIs of the documents affected by the change.

`presentation-index`

This web web service helps to maintain the external index used by the presentation system. It works in exactly the same way as **`index`** except that it does not log updates to staged content items, since staged content items (unpublished revisions of published content items) should not be visible to web site visitors.

indexer web application

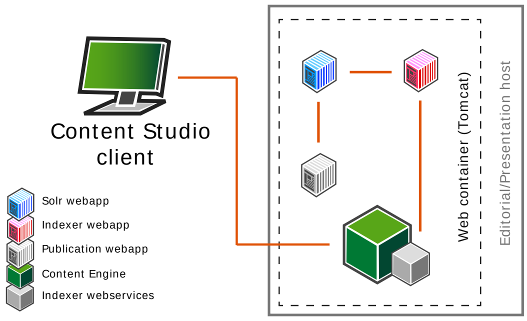
An **`indexer`** web application runs inside an application server together with a **`solr`** web application. Every five seconds, it submits a requests to one of the indexer web services and obtains the URIs of the documents that have changed in the last 5 seconds. It then submits requests to the Content Engine for these documents, passes them through an XSL filter to prepare them for indexing and posts the results to **`solr`**.

solr

`solr` also runs inside an application server. It generates and maintains an index based on the documents submitted by its **`indexer`**. It also responds to any search requests submitted to it, either from Content Studio clients or from publication web applications.

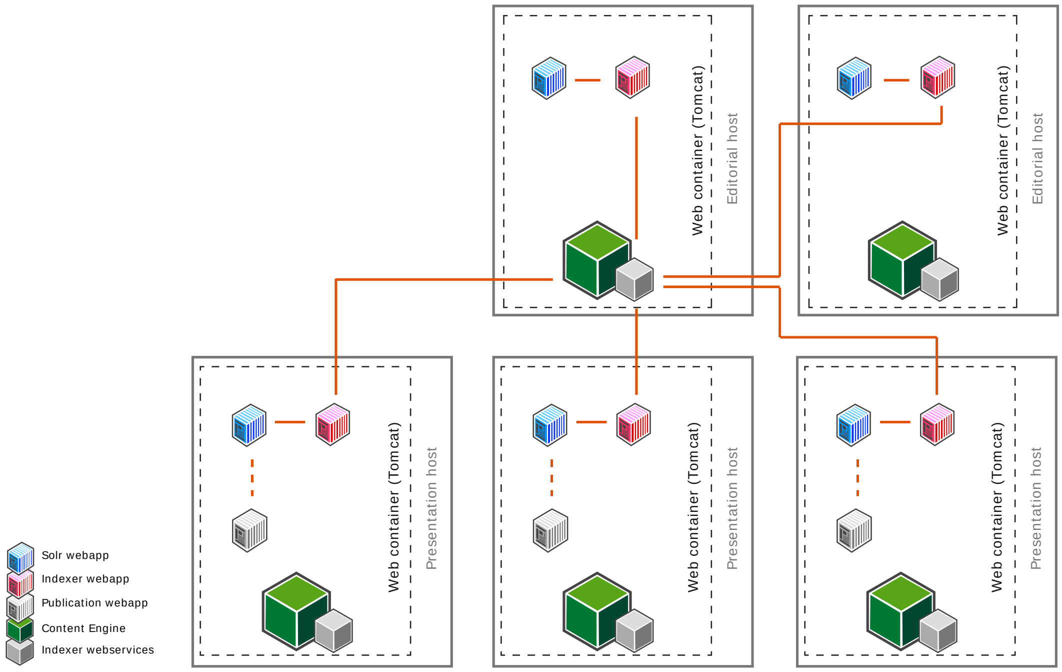
5.1 The Standard Configurations

In a standard Content Engine installation, both **`solr`** and the **`indexer`** application are deployed alongside the Content Engine in the same Tomcat instance. The **`solr`** instance is used to provide search functionality for Content Studio. Template developers can optionally use the same **`solr`** instance to provide search functionality for their publication web applications (although you should never do this for production purposes). The following illustration shows a single-host installation of the Content Engine set up in this way:



This kind of configuration is not recommended for production purposes since the indexing requirements for publication web applications are very different from Content Studio's requirements (see section 5.2).

In a multiple-host installation, the hosts on which the Content Engine runs are typically specialized: some are **editorial host**s, supporting a network of Content Studio clients, while others are **presentation host**s supporting public access to the organization's publications. The default configuration of the search components (as described in the Escenic Content Engine Installation Guide) is, however, almost the same:



The differences between the two configurations are:

- The indexer web applications on the editorial hosts are set up to use the internal indexer web service (**`index`**), while the indexer web applications on the presentation hosts are set up to use the external indexer web service (**`presentation-index`**).

- Only one instance of each indexer web service is used, for reasons of efficiency. Using the indexer web services in every Content Engine can result in a lot of unnecessary database accesses.

The web service used by each **`indexer`** web application is specified by means of an **`Environment`** element in the Tomcat **`context.xml`** file, as described in Install Application Server.

5.2 Modifying The Standard Configuration

The standard search configuration works well enough for development and test purposes, but is not suitable for a production environment. This section discusses some of the kinds of changes you can make, and some of the issues involved.

5.2.1 Using the Right Indexer Web Service

The Content Engine provides two indexer web services, one for internal use (called **`index`**) that logs information about all changes made to content items, and one for external use (called **`presentation-index`**) that omits changes made to staged content items. However, the standard search configuration includes only one **`solr`** instance and one indexer webapp, which are configured to use the internal web service. This means that if you use the standard configuration for production purposes, public search results will contain results from staged content items that are not themselves public.

You can avoid this problem in two ways:

Disable content item staging
: This may be an acceptable solution in some cases, but will result in reduced functionality for writers and editors. See Content Item Staging for details.

Configure a second `solr` instance and indexer webapp
: One **`solr`** instance/indexer webapp is configured to use the **`index`** web service, and the other is configured to use the **`presentation-index`** web service.

5.2.2 Customizing the Index Schema

The default **`solr`** index schema delivered with the Content Engine is optimized for editorial purposes: it indexes all the fields needed to support the search functionality provided by Content Studio, resulting in very large indexes. This is acceptable in the editorial context, since the number of concurrent Content Studio users, even in a very large organisation, is not likely to be very large. The **presentation host**s in a large Escenic installation, however, can be required to serve many thousands of concurrent users, and the default **`solr`** configuration may perform poorly in this context.

In other words, the default configuration is fine for the **editorial host**s in a production system, but for the **presentation host**s you are recommended create a custom indexer configuration that only indexes the fields actually needed to support the kinds of search required in your publications.

To do this, open **`var/lib/escenic/schema.xml`** for editing on each of your **presentation host**s, and modify the index schema to meet your requirements. Editing this file is outside the scope of this manual. In order to tune the search engine you need to take account of both the contents of your publications, your users' needs with regards to search and the limitations imposed by your particular hardware configuration. For further information and advice on tuning, see the Solr documentation on http://lucene.apache.org/solr/.

5.2.3 Isolating The Search Engine

Searching and indexing can be resource-intensive processes. Co-locating **`solr`** with the Content Engine can therefore sometimes prove to be a bad idea, especially in the case of **presentation host**s, which may need to respond to large numbers of simultaneous searches and ordinary document requests. However, since the Content Engine, **`solr`** and the **`indexer`** are all independent web applications that communicate via standard, stateless HTTP requests, you can locate them wherever you want in order to achieve the best possible load distribution.

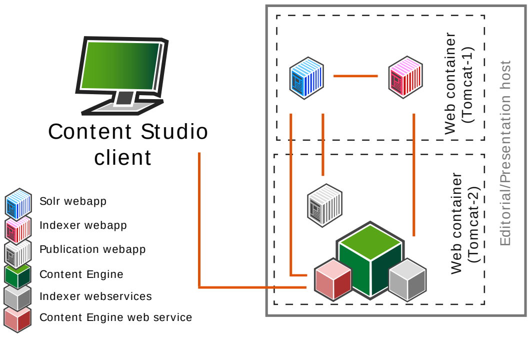
The following sections describe two different ways of isolating the search engine:

- Running the search engine in a separate webapp container.
- Running the search engine on a separate host.

For a production system you should **never** use the default configuration where **`solr`** runs in the same webapp container as the Content Engine. The reason for this is that **`solr`** can at times consume large amounts of memory and trigger large garbage collection operations in the JVM, which has severe effects on Content Engine performance. At production installations, **`solr`** **must** be run in a separate JVM from the Content Engine if you don't want to run into unnecessary performance problems. The simplest way to achieve this is to run it in a separate webapp container (that is, a separate Tomcat instance) as described in section 5.2.3.1.

5.2.3.1 Search Engine in Separate Container

The following illustration shows a single-host installation where **`solr`** is running in a separate webapp container:



To do this you would need to:

1. Install a second Tomcat instance on your host. Make sure you set it up to listen on another port than your main Tomcat instance.
2. Remove the **`solr`** and **`indexer`** web applications supplied with the Content Engine from your original Tomcat instance.
3. Deploy the **`solr`** and **`indexer`** web applications supplied with the Content Engine on the new Tomcat instance.
4. Add the following **`Environment`** elements to your new Tomcat instance's **`context.xml`** configuration file:

```
<Environment name="escenic/indexer-webservice"
             value="http://localhost:8080/indexer-webservice/index/"
             type="java.lang.String" override="false"/>
<Environment name="escenic/index-update-uri"
```

```
              value="http://localhost:8081/solr/update/"
              type="java.lang.String" override="false"/>
<Environment name="escenic/solr-base-uri"
              value="http://localhost:8081/solr/"
              type="java.lang.String" override="false"/>
<Environment name="escenic/head-tail-storage-file"
              value="/opt/escenic/indexer/head-tail.index"
              type="java.lang.String" override="false"/>
<Environment name="escenic/failing-documents-storage-file"
              value="/opt/escenic/indexer/failures.index"
              type="java.lang.String" override="false"/>
```

This sets up the **indexer** web application to use the indexer web service on the original Tomcat instance (port 8080 in this example) and the **solr** installation on the new Tomcat instance (port 8081 in this example).

5. Modify your Content Engine configuration to use the new **solr** installation. To do this you need to edit *configuration-layer-root***/com/escenic/webservice/search/DelegatingSearchEngine.properties** and set the **solrURI** property as follows:

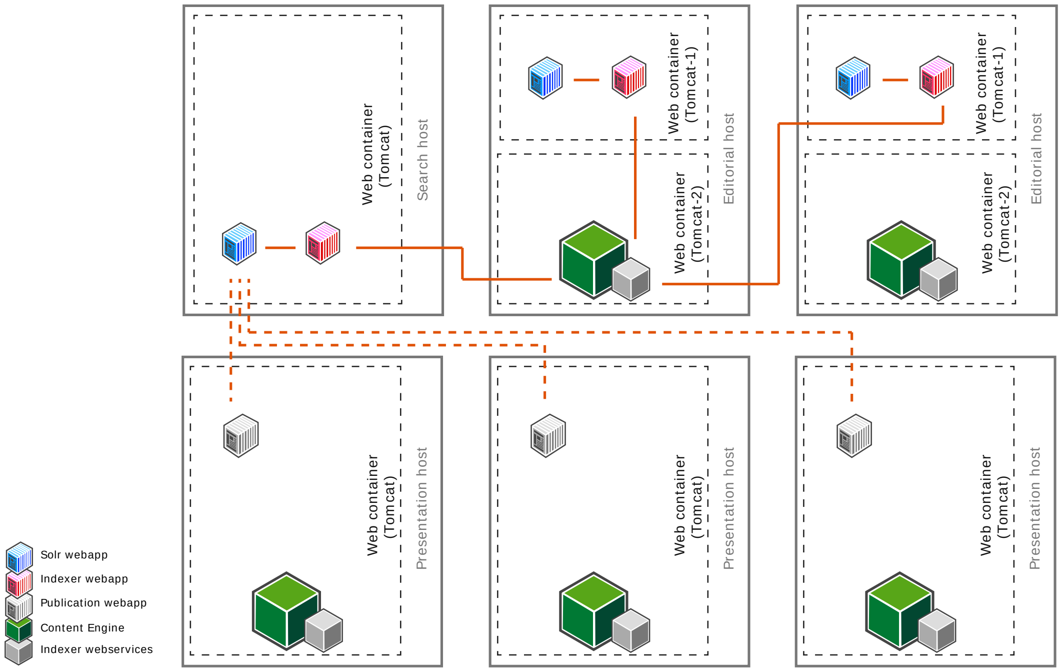
```
solrURI=http://pub1.example.com:8081/solr/select
```

(assuming your new Tomcat instance is listening on port 8081).

Isolating **solr** in this way would ensure that it does not have too severe an effect on the operation of the Content Engine. Ultimately, of course, performance is limited by the hardware the installation is running on, but separating **solr** from the Content Engine in this way will avoid a major cause of unnecessary performance degradation. If **solr** activity still causes performance problems, then you should consider moving **solr** to a different host as described in section 5.2.3.2.

5.2.3.2 Search Engine on Separate Host

The following illustration shows a multi-host installation where **`solr`** is running in a single, dedicated search host:



To do this you would need to:

- Install Tomcat on your search host.
- Deploy the **`solr`** and **`indexer`** web applications supplied with the Content Engine on the search host.
- Copy the **`solr`** configuration files supplied with the Content Engine to the search host, making sure to modify the index schema to meet your requirements, as described in section 5.2.2.
- Modify your publication web applications to use the **`solr`** instance on your search host.

Isolating **`solr`** in this way would ensure that re-indexing, for example, does not adversely affect response times on your **presentation host**s. However, it would also make the search host a single point of failure. A more robust solution would be to have two or more search hosts, and direct requests to them via a load balancing and/or fail-over service so that:

- Requests are evenly distributed between the search hosts
- If one host fails, requests are re-directed to other hosts

Load balancing/fail-over strategies can be implemented in many different ways using a variety of different standard products and technologies. Exactly how you do this is outside the scope of this manual: the point is that since all the components involved in searching and indexing communicate via standard, stateless HTTP requests, you can do it using standard web management techniques.

5.2.3.3 Setting Timeouts and Limits

You can control Indexer performance by setting various timeouts and limits. These limits are set in two different places:

- The Tomcat **`context.xml`** file
- In your configuration layers

You can set the limits by adding the following **`Environment`** elements to the Tomcat **`context.xml`** file:

`escenic/index-producer-connection-timeout`
: The connection timeout for connection to the indexer web service, specified in milliseconds. The default value is 4000.

`escenic/index-producer-socket-timeout`
: The socket timeout for connection to the indexer web service, specified in milliseconds. The default value is 4000.

`escenic/index-consumer-connection-timeout`
: The connection timeout for connection to Solr, specified in milliseconds. The default value is 4000.

`escenic/index-consumer-socket-timeout`
: The socket timeout for connection to Solr, specified in milliseconds. The default value is 4000.

You can parameters governing the indexing of binary files by editing **`com/escenic/search/index/BinaryIndexerPlugin.properties`** in the default configuration layer. Of particular interest are the properties

`maxWaitTime`
: The maximum time to wait for indexing of a binary file, specified in milliseconds. The default value is 3000.

`maxFileSize`
: The maximum size of binary file to index, specified in bytes. The default value is 20971520 (i.e, 20MB).

5.3 Re-indexing

From time to time it may be necessary to completely re-generate an index. Reasons for re-indexing include:

- A Content Engine upgrade. Some upgrades include modifications to the default **`solr`** schema used by Content Studio.
- Changes to one or more of your publications, or the addition of new search functionality require changes to your own custom **`solr`** schema.

In theory, all you need to do to re-index your publications is click on the **Reindex...** button on the **`indexer`** web application's admin page. However, the re-indexing process may take several hours on large sites, and while it is in progress, search requests will return incomplete results. In many production environments, reduced search functionality over several hours is not acceptable. In such cases you can avoid the problem by generating the new index using a separate, non-production Tomcat instance, and then copying the new index to the production environment.

The exact procedure for doing this is installation-dependent, but involves the following general steps:

1. Install a Tomcat instance somewhere in your network that you can use for generating the new index. (Alternatively you can make use of some existing non-production Tomcat instance - on a test server, for example.)
2. Deploy **`solr`** and **`indexer`** web applications on the server.
3. Copy **`context.xml`** from one of your production Tomcat configurations to your indexing Tomcat instance. This ensures that your **`indexer`** web application will be correctly configured to communicate with the Content Engine's indexer web service. By default, **`context.xml`** is located in **`/opt/tomcat/conf/`**.
4. Copy the **`solr`** configuration files (usually located in **`/var/lib/escenic/solr/default/`**) from your production **`solr`** instance to your indexing instance.
5. Modify the copied configuration as necessary for generating the new index. You might, for example, need to replace the schema file, **`schema.xml`**.
6. Start a browser and display the new **`indexer`** web application's admin page (**`http://`***host*:*port***`/indexer-webapp/admin/`**)
7. Click on **Reindex...**, then click on your browser's **Back** button to redisplay the admin page.
8. Wait for the indexing job to complete. The **Current state** section of the admin page shows the progress of the indexing operation, but it is not refreshed automatically. Click on your browser's **Refresh** button from time to time and check the **Number of documents read but not yet processed** value. When this value reaches 0, indexing is complete.
9. Test the generated index. The easiest way to do this is to use Solr's administration interface. Open a web browser, go to **`http://`***host*:*port***`/solr`** and follow links to the correct administration page (exactly how you get there is installation-dependent). The administration page contains a search field that you can use to execute test searches, plus links to the Solr documentation.
10. If you are not satisfied with the results, make the required changes to your configuration files, and try again (from step 6). Otherwise, continue.
11. Stop the Tomcat instance in which your production **`solr`** instance is running.
12. Copy your modified **`solr`** configuration files from your indexing instance to the production instance.
13. Copy the new index file (usually **`/opt/escenic/indexer/head-tail.index`**) from your indexing instance to the production instance.
14. Restart your production Tomcat instance.

6 Caching

In order to reduce the load on the database, the Content Engine maintains a number of internal caches. Objects and other items of information loaded from the database are cached in memory, and may in fact be cached in more than one of the caches. Once an item has been added to a cache, it is retained until:

The item is modified
: Any item that is modified is automatically deleted from the cache.

The cache is full
: The cache has a maximum size. When the cache reaches this maximum size, some items are deleted from the cache to make room for the new arrivals. The least-recently used items are deleted.

The cache is manually flushed
: The caches can be manually flushed using **`escenic-admin`**. See section 6.1 for details.

The server is shut down
: Whenever the server is shut down or restarted, all caches are flushed.

A typical example of a cache configuration file, would look like this:

```
maxSize=1000
validSeconds=-1
throwCount=300
objectLimit=10000
objectsToKill=100
```

6.1 Flushing Caches

Caches can be flushed while the server is running. To do so:

1. Go to the **`escenic-admin`** application's **component browser**. (For details, see section 2.1.15).
2. Use the component browser to find the cache component you want to reset.
3. Invoke the cache component's **`flush`** method. For instructions on how to do this, see section 2.1.15.2.

6.2 Tuning The Object Caches

You can tune the object caches by setting the following cache properties:

`maxSize`
: The maximum number of objects allowed in the cache.

`throwCount`
: The number of objects that will be removed from the cache once **`maxSize`** is reached. You should normally set it to at least 10% of **`maxSize`**, since the process of identifying which objects to remove carries an overhead. If you set **`throwCount`** very low, then a small number of objects

will be removed very frequently. Removing a larger number of objects less often is usually more efficient.

validSeconds
You can use this property to set a time threshold (in seconds) after which objects are removed from the cache. This prevents modified objects from surviving in the cache too long. However, the Content Engine is in general efficient at removing invalid objects, so it can usually be set to **-1** (which disables this process).

These properties are set by editing configuration files. For general information about editing Escenic configuration files, see section 4.2. You can also make temporary changes to cache settings while the Content Engine is running using the **escenic-admin** application's component browser (see section 2.1.15).

Ideally, all caches should be large enough to hold all the elements ever added to it: this would mean an element would never need to be loaded from the database more than once. In practice, this is unlikely to be possible due to memory limitations, so trade-offs must be made. Tuning the object caches is therefore usually a trial-and-error process aimed at finding the best possible set of trade-offs for a particular installation. If cache limits are set too low, the database will be accessed too often, resulting in reduced performance. If cache limits are set too high, memory can be overloaded, which also results in reduced performance. It is worth noting, however, that a high cache limit can only cause problems if the cache space is actually used: setting a cache limit too low, however, is guaranteed to have some effect on performance.

In general, the best way to tune the caches is to regularly check the performance summary displayed on the **escenic-admin** application's **Performance Summary** page (see section 2.1.4). This summary contains a general **Caches** section for the Content Engine's caches, plus individual sections listing information about the caches in each web application. For information on how to interpret the statistics displayed in these tables, see section 2.1.4.1.

When determining cache sizes you also need to take into account how much memory they will occupy, and this is a function of both the number of objects in the cache **and** the size of those objects. This is particularly significant in the case of web applications' **PresentationArticleCache**s, since the size of the objects held in them can vary widely. If a publication's typical content items are large, then its **PresentationArticleCache** may become very large. If you know the average size of the articles in a publication, then you can estimate the memory the cache is likely to consume as follows:

If an 'average' document is 15KB of plain (8-bit) text (either HTML or XML), it will basically occupy 30KB as a Java object because Java uses 16-bit encoding internally. In addition, there is a fixed overhead of around 5KB per article, giving a total memory requirement of around 35KB. So if you set the **PresentationArticleCache**'s **maxSize** property to 10000 documents, the cache may require up to 350 MB of memory.

The following sections contain some basic items of useful information about each of the object caches listed on the **escenic-admin** **Performance Summary** page. The following information is provided about each cache:

- The cache component name. This is the name you use to locate the cache in the component browser (although the easiest way to find it is just to click the cache's link on the **escenic-admin** **Performance Summary** page).
- The cache configuration file name. This is the file you need to create or edit to make permanent changes to the cache configuration. Global Content Engine cache configuration files may be added to one or more of your configuration layers. For information about configuration layers, see section

4.3. Web application cache configuration files must be added to the web application's **WEB-INF/localconfig** folder.

- Typical object size. You need this to work out how much memory the cache will use when it is full.

6.2.1 Global Caches

The caches described in the following sections are the global caches displayed on the **escenic-admin** **Performance Summary** page. Other global caches may appear on this page if plug-ins have been installed.

6.2.1.1 AgreementCache

Cache component name

/neo/io/content/cache/AgreementCache

Cache configuration file

configuration-layer-root/**neo/io/content/cache/AgreementCache.properties**

Typical Average Object Size

1Kb.

6.2.1.2 ArticleListCache

Cache component name

/neo/io/content/cache/ArticleListCache

Cache configuration file

configuration-layer-root/**neo/io/content/cache/ArticleListCache.properties**

Typical Average Object Size

1Kb.

6.2.1.3 ArticleSourceMap

Cache component name

/neo/io/content/cache/ArticleSourceMap

Cache configuration file

configuration-layer-root/**neo/io/content/cache/ArticleSourceMap.properties**

Typical Average Object Size

1Kb.

6.2.1.4 ArticleXmlCache

> This cache is not used.

6.2.1.5 CatalogCache

Cache component name

`/neo/io/content/cache/CatalogCache`

Cache configuration file

configuration-layer-root`/neo/io/content/cache/CatalogCache.properties`

Typical Average Object Size

1Kb.

6.2.1.6 ExternalContentCache

Cache component name

`/neo/io/content/cache/ExternalContentCache`

Cache configuration file

configuration-layer-root`/neo/io/content/cache/ExternalContentCache.properties`

Typical Average Object Size

1Kb.

6.2.1.7 LayoutCache

Cache component name

`/neo/io/content/cache/LayoutCache`

Cache configuration file

configuration-layer-root`/neo/io/content/cache/LayoutCache.properties`

Typical Average Object Size

1Kb.

6.2.1.8 ObjectCache

Cache component name

`/io/api/ObjectCache`

Cache configuration file

configuration-layer-root`/io/api/ObjectCache.properties`

Typical Average Object Size

1Kb.

6.2.1.9 PublicationCache

> This cache's `maxSize` should be set to a large enough value to ensure that it never needs to be flushed. (that is, large enough to hold references to all sections of all publications).

Cache component name

`/neo/io/content/cache/PublicationAttributeCache`

Cache configuration file

configuration-layer-root`/neo/io/content/cache/PublicationAttributeCache.properties`

Typical Average Object Size

1Kb.

6.2.1.10 ReferenceEntityCache

Cache component name

`/neo/io/content/cache/ReferenceEntityCache`

Cache configuration file

configuration-layer-root`/neo/io/content/cache/ReferenceEntityCache.properties`

Typical Average Object Size

1Kb.

6.2.1.11 RelationshipCache

Cache component name

`/io/api/RelationshipCache`

Cache configuration file

configuration-layer-root`/io/api/RelationshipCache.properties`

Typical Average Object Size

1Kb.

6.2.1.12 SectionCache

> This cache's **`maxSize`** should be set to a large enough value to ensure that it never needs to be flushed (that is, large enough to hold references to all sections of all publications).

Cache component name

`/neo/io/content/cache/SectionCache`

Cache configuration file

configuration-layer-root/**`neo/io/content/cache/SectionCache.properties`**

Typical Average Object Size

1Kb.

6.2.1.13 SectionParameterCache

> Section parameter caching can be disabled by setting the property **`parameterCache`** to 'false' in **`neo/io/managers/SectionManager.properties`**. In production this property should aways be set to true, which is the default. This property should set to be **`false`** in template development environments, like this:
>
> ```
> parameterCache=false
> ```

Cache component name

`/neo/io/content/cache/SectionParameterCache`

Cache configuration file

configuration-layer-root/**`neo/io/content/cache/SectionParameterCache.properties`**

Typical Average Object Size

1Kb.

6.2.1.14 SectionSourceMap

Cache component name

`/neo/io/content/cache/SectionSourceMap`

Cache configuration file

configuration-layer-root/**`neo/io/content/cache/SectionSourceMap.properties`**

Typical Average Object Size

1Kb.

6.2.2 Web Application Caches

6.2.2.1 PresentationArticleCache

Cache component name

`/neo/xredsys/presentation/cache/PresentationArticleCache`

Cache configuration file

webapp/**`WEB-INF/localconfig/neo/xredsys/presentation/cache/PresentationArticleCache.properties`**

Typical Average Object Size

Very variable, very publication dependent, but often somewhere between 20 and 40Kb.

6.2.2.2 PresentationListCache

Cache component name

`/neo/xredsys/presentation/cache/PresentationListCache`

Cache configuration file

configuration-layer-root/**`neo/xredsys/presentation/cache/PresentationListCache.properties`**

Typical Average Object Size

1Kb.

6.2.2.3 PresentationPoolCache

> This cache's **`maxSize`** should be set to a large enough value to ensure that it never needs to be flushed.

Cache component name

`/neo/xredsys/presentation/cache/PresentationPoolCache`

Cache configuration file

configuration-layer-root/**`neo/xredsys/presentation/cache/PresentationPoolCache.properties`**

Typical Average Object Size

1Kb.

6.2.2.4 PresentationSectionCache

> This cache's **`maxSize`** should be set to a large enough value to ensure that it never needs to be flushed.

Cache component name

/neo/xredsys/presentation/cache/PresentationSectionCache

Cache configuration file

configuration-layer-root/**neo/xredsys/presentation/cache/PresentationSectionCache.properties**

Typical Average Object Size

1Kb.

6.3 Distributed Caching

In a multi-server installation, each server running the Content Engine has its own set of caches, and all these caches must be synchronized with each other to some extent. Specifically, whenever a change is made that can potentially cause an item in a cache to become invalid, that change must be reported to all servers, so that the appropriate caches can be checked and the invalid item can be removed, if necessary. The basic mechanism is that the Content Engine generates an event each time a potentially cache-invalidating change is made. At the same time, the Content Engine also listens for such events generated by other Content Engine instances, and when it receives such an event, checks the appropriate cache and if necessary, removes the invalid item.

There are, however, two ways to set up distributed caching:

In a typical multi-server installation, different servers have different functions. There are two basic server types:

Publishing servers
: A publishing server is a 'back-end' server used by editorial staff to create and modify publication content using Content Studio.

Presentation servers
: A presentation server is a 'front-end' server used to serve publication content.

As a general rule, therefore, a publishing server is a change-generating server, and a presentation server is not. This is, however, not always the case, since some publications include functionality that enables "reader participation" of one kind or another. If the Forum plug-in is installed, for example, then presentation servers will also be change-generating servers.

> For multi-server setup, you should make sure to set the **escenic.server** system property on all your Content Engine instances. Each Content Engine instance should have this property set to its own host name or IP address.

6.3.1 EventManager Service

This service is responsible for the communication among different escenic content engine servers. You can find it here,

configuration-layer-root/**io/api/DatabaseEventManager**.

This page also lists some performance metrics along with the properties of this component. If this service does not run properly then your multi-server setup will not work. You can check the health of this service from **Home** > **View Services** page in **`escenic-admin`**.

6.3.1.1 Standalone Database

It is possible to use a different database as the **`EventManager`**. What you need to do is to create another **`ContentManager`** with different read and update connectors, collectors and throttle services and then set this **`ContentManager`** as the **`EventManager`**.

6.4 Cache Validation

In certain circumstances, the Content Engine's internal caches can be updated with an outdated copy of an object, resulting in stale content in the caches. You can prevent this problem ever arising by setting a configuration property called **`validateObjectsAfterInsert`**. Setting this property to **`true`** causes the Content Engine to validate all newly-cached items with the database, thereby ensuring that the caches will never contain stale information. The property is set to **`false`** by default, since database validation of every cache insertion is a potentially time-consuming operation and may have a significant effect on performance. You can, however, set the property if the possibility of stale content appearing in the caches is unacceptable.

To set this property, add *configuration-layer-root*`/io/api/CacheManager.properties` to one of your configuration layers and include the required setting in the file:

```
validateObjectsAfterInsert=true
```


7 Bootstrapping

By default, when the Content Engine is started, all its caches are empty. In a test or development environment, where activity is usually very low, this is not a problem. For a production system running a busy site, however, the level of requests can be so high as to completely cripple the site if all requests have to be fully processed rather than served from the cache. For this reason, the Content Engine includes an **`InitialBootstrapper`** component that can be used to protect the Content Engine from traffic during start-up, allowing it to prime the caches with frequently-requested pages before it is required to respond to real requests.

The **`InitialBootstrapper`** component works by:

- Intercepting incoming requests and returning HTTP 503 responses (Service Unavailable).
- Simultaneously submitting a series of dummy requests for frequently requested pages, thereby priming the caches with content that will enable fast responses to many requests when the bootstrap sequence is completed.

Bootstrapping is initialized on a per-publication basis by setting the bootstrapOnStartup parameter in each publication's **`feature`** resource. The **`bootstrapOnStartup`** parameter allows you to specify the individual sections of a publication that are to be bootstrapped.

Details of how the **`InitialBootstrapper`** component carries out the bootstrap operation can be controlled by setting properties in the *configuration-layer-root*/**`neo/io/content/InitialBootstrapper.properties`** configuration file, described in the following section.

7.1 InitialBootstrapper

`InitialBootstrapper` inherits properties from:

- **`java.lang.Object`**

It also has the following properties of its own:

`secondsToWait` **(read/write)**
int

The number of seconds that the InitialBootstrapper should wait before trying to load the publications. Note that this time should include the time it takes from Escenic components loading to the application server being ready and accepting requests. If this value is too low, then requests may be stopped by the server, and the component will fail. If this value is too high, then the startup time of Escenic might appear to be longer than nessecary. It is by default set to wait 60 seconds. It is better that this value is too high rather than too low.

`timeoutSeconds` **(read/write)**
int

The number of seconds to try retrieving a publication. By default, if a publication has not finished bootstrapped within 30 seconds, it will continue to the next publication.

`threadCount` (read/write)
int
the number of simultaneous threads to use when bootstrapping. Typically this should be set to the same number of processors

`articlesToRetrieve` (read/write)
int
The number of articles to retrieve from the front page. Typically, the default value of "1" is satisfactory. The bootstrapper will keep trying to retrieve articles until it successfully loads this number of articles from the front-page.

`articlesToAttempt` (read/write)
int
The number of articles to attempt to retrieve from the front page. Typically, the default value of "5" is satisfactory. This means that after 5 failed attempts it will stop trying to retrieve articles from the section in question, and move on to the next.

`depth` (read/write)
int
The default depth to try to probe when going throught the section tree. By default, a publication's top section along with its children are probed, i.e. the depth is set to 2. Setting this property has effect when the bootstrapOnStartup is set to the keyword true. This value can be overridden on a per-publication basis, by specifying a number in the bootstrapOnStartup feature.

`failureThreshold` (read/write)
int
The number of failures that are to be tolerated in a publication. By default, the bootstrapper will stop accessing a publication if it fails 5 sections.

`token` (read-only)
String
The value of the token that the initial bootstrapper will use as a query parameter when issuing the HTTP requests.

`bootstrappedPublications` (read-only)
String
A list of publications that were bootstrapped when the bootstrapper was run.

`bootstrapped` (read/write)
boolean
Wether or not all publications have been bootstrapped. This value may be set to true before or during bootstrapping, and any running bootstrap threads will stop their work. This property must be false in order for bootstrapping to start. When bootstrapping is finished, this property is automatically set to true. By default, this property is false upon startup, and after bootstrapping, will be true.

`threadRunning` (read-only)
boolean
true if any bootstrapping is happening right now, false otherwise. Simply an indicator of wether or not the bootstrapper is active.

8 Throttling

The Content Engine has a a number of throttle services that you can use to limit the number of concurrent requests that various parts of the system will attempt to handle. Once the specified threshold is reached, requests to the overloaded part of the system will be refused.

The following throttle services are available:

`WebServiceThrottle`
: Limits access to the Content Engine web service used by Content Studio.

`DatabaseUpdateThrottleService`
: Limits the number of concurrent database updates.

`DatabaseReadThrottleService`
: Limits the number of concurrent database reads.

`JspThrottleService`
: Limits the number of concurrent page requests.

The throttle services are all enabled by default and set up with default configurations. You should **not** switch the throttle services off in a production environment, as overload situations are then likely to be handled in an unpredictable manner. You can, however, configure the throttle services by editing the appropriate files in one of your configuration layers (see chapter 4).

All the throttle services are instances of the **`ResourceThrottle`** class, and are configured by setting **`ResourceThrottle`** properties. The most important property you can set is **`maximumConcurrent`**, which determines the maximum number of concurrent requests that will be handled.

For **`WebServiceThrottle`**, **`DatabaseUpdateThrottleService`** and **`DatabaseReadThrottleService`**, **`maximumConcurrent`** is set by default to 100, which is a relatively high value that can most likely be left unmodified. Database accesses should normally be controlled by the database system itself, so **`DatabaseUpdateThrottleService`** and **`DatabaseReadThrottleService`** can be seen as "failsafe" devices that will only ever be needed if something is badly configured elsewhere. Similarly, usage of the Content Engine's web service is unlikely under normal operation ever to reach a level of 100 concurrent accesses, even in large installations, so if this limit is ever reached, it is probably a sign that something is wrong.

`JspThrottleService`, on the other hand, is not just a failsafe device, it is vital to ensuring that the Content Engine handles periods of high activity in a controlled manner. Moreover, the optimum setting for **`maximumConcurrent`** is entirely installation-dependent, and must be based on experience and testing. For this reason, the default value is deliberately set set to a low value of 10. There is no sensible default: you must observe the Content Engine's performance and arrive at the optimum setting by trial and error.

In order to find out the optimum settings in a production environment, you need to examine performance numbers, and the number of HTTP 503 messages returned. The **`escenic-admin`** application's **Performance summary** option displays a page of performance data including an **Activity Monitors** section containing throttle activity data (see section 2.1.4.3).

The **Current Usage** column in the **Activity Monitors** section shows the current number of concurrent accesses. Above the **Current Usage** section, the **`/neo/io/reports/HitCollector`** entry in the **`Load Averages`** section shows the request load reaching the Content Engine. The

Failures field shows how many requests have failed or been rejected. If failures are being recorded by the `/neo/io/reports/HitCollector`, and you see that incrementations of this value coincide with high **Current Usage** values for the `JspThrottleService`, then `maximumConcurrent` is probably set too low.

All the throttles are implemented using the `ResourceThrottle` class, and therefore have the same set of configuration properties, described in the following section.

8.1 ResourceThrottle

`ResourceThrottle` inherits properties from:

- `java.lang.Object`

It also has the following properties of its own:

`maximumConcurrent` (read/write)
int
The maximum number of concurrent usages of a specific resource. This number decides how many simultaneous clients can use the resources at a time.

`availableCapacity` (read-only)
int
The number of free resources that this throttle attempts to govern. This number changes every time someone checks in a resource, or the maximumConcurrent value changes.

`overloadMessage` (read/write)
String
The message that clients can use when handling the case in which the server has been overloaded. The hard-coded default message is "Resources Exhausted".

`activeResources` (read-only)
Collection
A list of string representations of all active resources. If a resource has become unavailable for a prolonged period of time, this will show what the resource is being used for.

`serviceRunning` (read-only)
boolean
Whether or not the service is running. This flag is modified by doStartService and doStopService.

`serviceEnabled` (read/write)
boolean
Whether or not the service is enabled. If the service is disabled, no log of activity will be kept, and no attempts to use resources (checkout) will fail.

8.2 Per-Publication Throttling

By default, the same throttle controls access to all publications. It may be, however, that you want to isolate the publications from one another, so that a traffic spike on one publication does not affect the performance of other publications. You can do this by defining additional throttle service components like the default `/neo/io/services/JspThrottleService` component. You can then:

- Configure different publications to use different throttle services.
- Set the **`maximumConcurrent`** property individually for each publication.

Note that doing this does not increase the total capacity of the server. If **`maximumConcurrent`** was already set to its optimum value in a single throttle set-up, then this number of concurrent requests must be shared out between the throttle services in the new set-up.

To set up additional throttle services:

1. Create a **`.properties`** file for each throttle service you want to create in one of your configuration layers. You might, for example, create a file called *configuration-layer-root*/**`throttles/MyThrottle.properties`**:
2. Add the following class definition.

   ```
   $class=neo.util.ResourceThrottle
   ```
3. Add the additional property settings you require. For example:

   ```
   maximumConcurrent=5
   ```
4. Since you've added new throttle services, you will probably need to reduce the **`maximumConcurrent`** setting of the default throttle service (**`/neo/io/services/JspThrottleService`**) accordingly. To do this, edit *configuration-layer-root*/**`neo/io/services/JspThrottleService.properties`**. (You may need to create this file if it does not already exist in the configuration layer.)
5. For every publication web application that is to use the new throttle service, you must edit the **`WEB-INF/web.xml`** file. Open the file, find the **`ECETimerFilter`** definition and add a new parameter definition as a child of the **`init-param`** element:

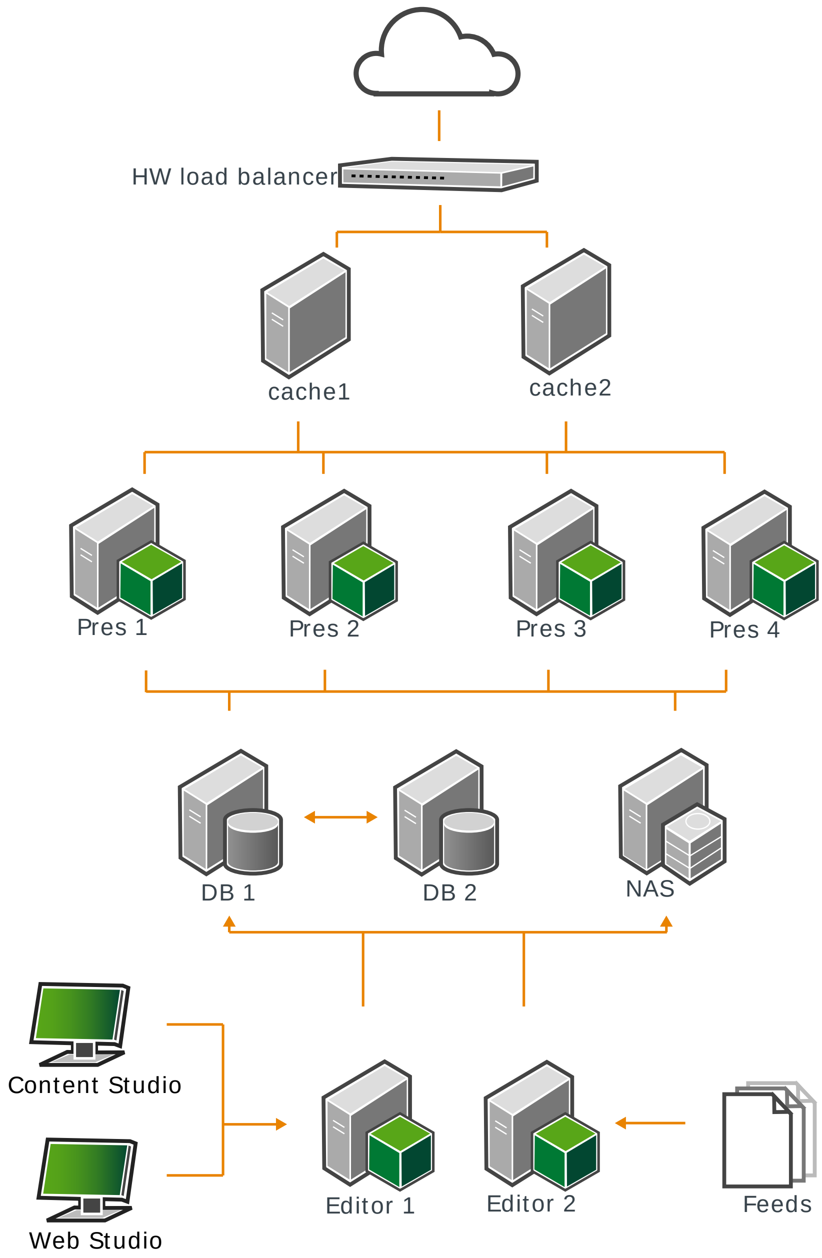
   ```
   <init-param>
     <param-name>throttle</param-name>
     <param-value>/throttles/MyThrottle</param-value>
   </init-param>
   ```

   The **`throttle`** parameter must be set to the name of the new throttle service (**`/throttles/MyThrottle`** in this case).

9 Performance

This chapter is intended to provide you with a starting point for identifying and solving the problems involved in ensuring that your Escenic site performs and scales well. The information it contains is general in nature, but wherever numbers are discussed, they are based on an assumption that the site will need to serve around 50 000 simultaneous users.

The architecture shown in the following diagram should cater for such numbers and includes all the components discussed in this chapter.



9.1 Scalability

Ensuring the scalability of a typical Escenic site is fundamentally a matter of correctly caching the content. It involves:

- Correctly tuning the Content Engine caches (see chapter 6).
- Running a distributed memory cache (**memcached**) to ease the load on the databases (see Distributed Memory Cache)
- Running a well-configured cache server, such as Akamai, Squid or Varnish in front of the application servers.

You will need:

- 6-8 engine hosts
- 2 database hosts
- Some kind of high availability solution for the file system (using HA proxy and virtual IPs, for example)
- 2-4 cache servers (multiplied by two if you are using Squid 2.x).

9.2 Web Server Set-up

The cache servers will in most cases also run a web server of some kind. Most of the advice given below is applicable in general terms whatever web server you use, but the specific examples are based on the Apache web server.

9.2.1 Web Server Tuning

Your web server needs to be tuned before going into production. The standard configuration included with the Apache distribution (or with our OS software package) is not optimised for high load web sites and you will therefore need to modify it. It is particularly important to configure the mpm_common worker module for production use. Be sure to read and understand the documentation for this module and then continue to these more general Apache performance guides:

- http://httpd.apache.org/docs/2.2/misc/perf-tuning.html
- http://www.devside.net/articles/apache-performance-tuning

> **Do not use** the prefork MPM worker, use the multi-threaded worker instead.

The Apache worker is set at compile time. Thus, if you have compiled it from source, check your build (**configure**) options to be sure the multi-threaded worker is selected. If you have installed Apache from RPM/DEB packages, you can usually use **rpm -qa | grep -i apache** or **dpkg -l "*apache*mpm"** to make sure that the high speed worker is being used.

This example shows how to configure the Apache worker for production use.

```
# worker MPM
<IfModule worker.c>
# We could increase ServerLimit to 64 and ThreadLimit/MaxClients to 8192,
# but be aware of the OOM of Death!!
```

```
# initial number of server processes to start
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html

#startservers
StartServers          3
ServerLimit           32

# minimum number of worker threads which are kept spare
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html

#minsparethreads
MinSpareThreads     512

# maximum number of worker threads which are kept spare
http://httpd.apache.org/docs/2.2/mod/mpm_common.html

#maxsparethreads
MaxSpareThreads     1024

# upper limit on the configurable number of threads per child process
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html

#threadlimit
ThreadLimit         4096

# maximum number of simultaneous client connections
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html

#maxclients
MaxClients         4096

# number of worker threads created by each child process
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html

#threadsperchild
ThreadsPerChild     128

# maximum number of requests a server process serves
# http://httpd.apache.org/docs/2.2/mod/mpm_common.html

#maxrequestsperchild
MaxRequestsPerChild  10000
</IfModule>
```

Make sure that you have a good understanding of the **MaxKeepAliveRequests** and **KeepAliveTimeout** parameters. The following values:

```
MaxKeepAliveRequests 1000
KeepAliveTimeout 5
```

work well in many production sites today. However, your needs may be different and you should therefore be careful when setting these parameters.

9.2.2 Why You Need a Web Server

It might seem tempting to remove the web server in order to simplify your server setup, especially since some cache servers (such as Varnish) offer powerful URL rewriting facilities, easy manipulation of HTTP headers and advanced access control lists.

However, production sites without a web server are rare, and if you plan to offer personalised sites (with user login, etc.), session binding is required. Some cache servers (such as Oracle Web Cache) have built-in session binding but others, such as Varnish, do not. Therefore, web servers are likely to be needed for the foreseeable future. For more on session binding, see section 9.8.1.

9.3 Database Performance

Database performance has an indirect impact on page rendering time and the responsiveness of the Content Engine as a whole. The effect of the database on overall performance is reduced by the Content Engine's caching strategy, but it is not eliminated. If a performance problem arises that appears to originate in the database, then it may be necessary to examine the database queries being executed in order to locate the "problem" SQL statements.

9.3.1 Identifying Slow Transactions

The Content Engine measures the time taken to execute every SQL statement. The **escenic-admin** application's **Performance summary** option (see section 2.1.4) displays a page of performance data that includes the average and peak access times for database engine queries and updates:

```
Database Engine Queries:
 Since last sample:
  2 db queries;
  effective 0.00Hz;
  average 4ms; peak 6ms;
  load 0.00 (delta -0.00);
  0 failures;
 Total:
  44 db queries;
  average 32ms.

Database Engine Updates:
 Since last sample:
  862 db transactions;
  effective 1.58Hz;
  average 2ms;
  peak 27ms;
  load 0.00 (delta -0.00);
  0 failures;
 Total:
  14148 db transactions;
  average 8ms.
```

These figures give you some idea of how the database is performing: a well-performing database will usually have an average access time of around 10 milliseconds for both queries and updates.

If a database operation takes more than 10 seconds (10,000 milliseconds), the Content Engine logs the transaction with an ERROR message in the log. The message contains information about the internal Content Engine transaction being performed, and may in some cases contain the actual SQL query being executed. If your database regularly has peaks of over 10 seconds, you should look in the log file to see what kinds of transactions are causing the problems.

The 10 second threshold for logging database transactions as errors is not fixed: you can set the threshold higher or lower by configuring the **/neo/io/managers/ContentManager** component.

To change the error threshold for read transactions, set the **readThreshold** property. To change the error threshold for write transactions, set the **updateThreshold** property.

You can reset these properties at run time using the **escenic-admin** application's **Component browser** option (see section 2.1.15). In this way you can easily set the properties to catch the peak access times currently being reported by the **Performance summary** option and find out what operations are causing the problems.

9.3.2 Troubleshooting Slow Transactions

If you find what looks like a particularly slow SQL transaction, you can configure the Content Engine to generate additional diagnostic information. To do this, use the **escenic-admin** logging level editor (see section 2.1.11) to set the logging category **com.escenic.sql.Logger** to one of the following values:

INFO
: Logs the SQL statements themselves before they are executed.

DEBUG
: Additionally logs the positional parameters of the prepared statements, as they are set.

You can now see all the SQL statements executed in the log, but you still don't know which particular statement is slow, nor do you necessarily know exactly how or why the individual statements come to be executed. You may have suspicions regarding some of the statements, however. You can set up the connection wrapper to dump the call stacks of these statements to the log. You should then be able to find from the stack traces which template files are responsible for the statements.

To generate stack dumps in this way you need to set the **/neo/io/connector/DebugConnection** component's **stackdumpRegExp** property to a regular expression that matches the SQL statement(s) you are interested in. If, for example, you are interested in all statements involving the **ArticleMetaContent** table, then you can set it to **/ArticleMetaContent/i** (the "**i**" at the end indicates that the expression is case insensitive). Then any SQL statement containing the string "articlemetacontent" will trigger a stack dump of the current thread to standard error.

> You can permanently set the logging level for **com.escenic.sql.Logger** by editing your **trace.properties** file (see chapter 11 for details).

9.3.3 Getting the Database to Scale

The real limitation governing the scalability of most read-heavy sites is the number of available database handles. Scaling up the application server layer does not make sense if the database can only deal with a limited number of read/write handles. Some high-end Oracle cluster solutions may possibly help solve this problem, but MySQL clusters cannot be used since they do not support sub-queries. Standard master/slave configurations are therefore the only option. As far as Escenic is aware, **all** current Content Engine sites are based on master/slave database configurations, regardless of what database they use.

> It is important to remember that both the read and write connection pools in ECE **must be configured to work on the master database instance**. The slave databases are for data redundancy (standby backup) only, and should not be used to serve requests as this **may** cause unforeseen behaviour.

You are recommended to install memcached on each of your **engine-host**s. **memcached** acts as a layer on top of the most important Content Engine cache, **/neo/xredsys/presentation/cache/PresentationArticleCache**, and significantly reduces the number of database read operations. See Distributed Memory Cache for details of how to install **memcached** on your **engine-host**s.

The relationship between memcached and in-memory caches

The Content Engine uses **memcached** as a "level 2" cache for the presentation layer. When the templates ask the presentation layer for an article, it first checks its in-memory cache - even if **memcached** is in use. If the object isn't found in the in-memory cache, then **memcached** is asked. If the object isn't available there either, then the object is loaded from the database and copied to the in-memory and **memcached** caches. When **memcached** is in use, some cache-related activities affect both the in-memory and **memcached** caches, while other activities affect only the in-memory cache. For example, functional activities such as adding and removing a specific item from a cache are propagated to **memcached**, whereas operational activities such as flushing the cache or setting the cache size, are not propagated to **memcached**.

9.3.4 Database Optimization

In order to maintain database performance levels, tables in which rows are frequently inserted and deleted need to be optimized at regular intervals. The Content Engine incorporates a service for this purpose, called **OptimizeTables**. The service is disabled by default. To enable it, add a file called *configuration-root*/**com/escenic/service/database/OptimizeTables.properties** to your common configuration layer, and set the following property in it:

```
serviceEnabled=true
```

By default, the service will then run at 05:05 each morning, and optimize the following tables:

ECELocks
ResourceLock
RemoteNotification

You can modify these (and other) defaults by adding further properties to the file. To see all available properties, examine the **OptimizeTables** service settings in the **escenic-admin** Component Browser (see section 2.1.15).

9.4 The TCP/IP Stack

The TCP/IP stack also imposes scalability limitations. How many simultaneous open TCP connections can your front-end servers handle, and how many open connections can be handled by the back-end components supporting them? Each layer in your software stack communicates with the layer below via TCP: load balancer -> cache server -> application server -> database/file system. There need to be sufficient connection handles available at each level to prevent bottlenecks occurring.

Each connection made to the load balancer results in a corresponding request to a cache server, so you need sufficient connection handles here to handle whatever maximum number of simultaneous requests you have decided upon. The cache servers should respond directly to a large number of requests, so you will need a much smaller number of connection handles between the cache servers and the application servers. Similarly, some requests will be responded to directly by the application server, so an even smaller number of connection handles is required for communication between the application server and the database/file system.

In order for your installation to perform well, the relationships between the number of connection handles available at each level in your server architecture must reflect the actual requirements of the traffic reaching your site.

9.4.1 Caching Servers

For the caching servers in the front layer of your server architecture you need have a clear understanding of TCP connection scalability issues.

The first thing you may notice as the load on your system increases, is that the cache server process runs out of file handles (unless its start script increases the right kernel parameter). This is because the operating system uses one file handle for each connection, and on many systems the default number of handles a single user process is allowed to create is 1024. This problem can be temporarily fixed with the **`ulimit -u`** command (on Linux and FreeBSD). To fix it more permanently you need to edit **`/etc/sysctl.conf`** (on Linux and FreeBSD) or **`/etc/system`** (on Solaris). You can set the maximum number of file handles up to several hundred thousand, so there is no real limitation here.

The operating system set up TCP connections between a local port and an anonymous port on the requesting host:

```
cache01:2323 -> otherhost:1237
```

Port numbers are defined in the TCP protocol as an unsigned 16-bit number which gives a maximum of 65535 ports. The local port number can, however, be re-used for connections to different hosts:

```
cache01:2323 -> otherhost:1237
cache01:2323 -> yetanotherhost:4545
```

This means that the maximum theoretical number of connections a cache server can handle is:

(65535 - *reserved-ports*) * *incoming-ip-addresses*

where *reserved-ports* is the number of ports reserved for system services by the operating system (usually 1024).

For this to work well, the load balancer in front of the cache must be transparent: that is, it must supply the IP address of the request source and not its own IP address.

For example, if three users are visiting your web site:

```
user1:2213 -> load-balancer:80 -> cache01:80
user2:1212 -> load-balancer:80 -> cache01:80
user3:5333 -> load-balancer:80 -> cache01:80
```

then ideally, **`cache01`** should see the IP addresses of the requesting clients (**`user1`**, **`user2`** and **`user3`**) rather than the IP of the load balancer. Your cache server will then be able to handle as many TCP connection as your load balancer can pass on (given that your operating system kernel manages to allocate and recycle enough TCP connections fast enough).

If this is not possible then an alternative (but less satisfactory solution) is to increase the maximum number of possible connections by adding additional interfaces (and corresponding IP addresses) to the load balancer and/or the cache server.

9.5 Searching with Solr

For guidance on how to scale the Solr search engine in a multi-host environment, see chapter 5.

9.6 Avoiding Single Points of Failure

A Content Engine's NFS server are potential single points of failure: if it goes down and you haven't done anything to prevent it, your web site will go down too. The only way to solve this problem is to duplicate these components: you have the same software installed on two hosts, but only run it on one of them, keeping the other ready as a backup. A **heartbeat** daemon (see http://haproxy.1wt.eu) is used to monitor the availability of the service and, if it goes down, start the service on the backup host.

This heart beat/fail over solution should also include a virtual IP address for the host running the critical service. All users of the service access it via the virtual IP address. If the service's primary host goes down and the heart beat starts the service on a backup host, the virtual IP address is moved from the primary host to the backup host. This ensures that no configuration changes are needed to any of the components using the service. Any components using the service at the time of failure will lose all current transactions and connections, but operation will resume on the backup host for any subsequent requests/transactions.

9.7 Optimizing the Operating System Kernel

A newly-installed operating system is not optimized for any particular use: its default settings are designed to cater for a wide range of different uses. For a server that is dedicated to performing a specific task, therefore, it makes sense to adjust the operating system's settings in order to maximize the performance of the software installed on it.

You can optimize the Linux kernel by editing **/etc/sysctl.conf**, and you can list the current kernel settings by entering:

```
# sysctl -a
```

You can find the names of all the possible kernel parameters you can set by browsing the **/proc/sys** tree in the file system. The kernel parameter **net.ipv6.route.max_size**, for example, corresponds to the file **/proc/sys/net/ipv6/route/max_size**.

For further information, see your operating system documentation, starting with the **sysctl** and **sysctl.conf** man pages.

Here is an example showing how to tune the Linux kernel (tested on 2.6.24) for running an Apache web server and Varnish cache server. Some of the settings here may in fact be redundant, but nevertheless, this configuration is known to work and has a proven track record of serving several high traffic web sites:

```
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
net.ipv4.tcp_fin_timeout = 3
net.ipv4.tcp_tw_recycle = 0
```

```
net.core.netdev_max_backlog = 30000
net.ipv4.tcp_no_metrics_save=1
net.core.somaxconn = 262144
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syn_retries = 2
```

9.8 Highly Interactive Sites

Highly interactive sites that incorporate social networking functionality, such as sites based on the Viz Community Expansion, have additional requirements. They can contain large amounts of user-generated information, and displayed pages frequently contain personalized and dynamic elements. it is therefore necessary to consider performance in the following additional areas:

- Session binding
- Edge Side Includes (ESI)

> If you are implementing a straightforward content-based site that does not offer large-scale user interaction, you can ignore this section.

9.8.1 Session Binding

For any Content Engine site that allows visitors to create user profiles and log in, you are recommended to make use of Apache's **`mod_proxy_balancer`** for providing sticky sessions and load balancing.

Be aware that you cannot use application server clustering (that is, sharing sessions between your application servers) since this requires that all objects written to the **`Session`** object are serializable. Currently, this requirement is not met by all Content Engine objects, and you therefore need to bind all sessions to one specific application server. You can either do this in your web server (for example, Apache's **`mod_proxy_balancer`**, as mentioned above) or alternatively in some cache servers, such as Oracle Web Cache.

9.8.2 Edge Side Includes

Edge Side Includes (ESI) is an XML-based language (and a W3C standard) that allows web page and template developers to include caching requirements in their page mark-up. This makes it possible to establish a differential caching policy that caches different parts of a page for different lengths of time. A page is essentially broken up into fragments with different caching policies. Some highly dynamic fragments (the number of messages in a user's inbox, for example) may be cached for a very short time or not at all, while parts that are likely to change less often (such as a news article or blog entry) can be cached for much longer. Big IP, Varnish, Akamai, Oracle Webcache and Squid 3 all support ESI.

The basic idea is that the application developer, who is the person best placed to know how long a given fragment should be cached, sends that information to the cache server in the form of ESI directives. With Varnish at least, no additional configuration is required to make the cache server respect ESI directives. This example shows how to set a cache time of one minute on a fragment.

```
<%@ taglib uri="http://jakarta.apache.org/taglibs/response-1.0" prefix="response"%>
```

```
<response:addHeader name="Cache-Control">
    s-maxage=60
</response:addHeader>
```

Template developers need to be aware that using ESI imposes constraints on how they structure their templates. They must also be sure to set the **s-maxage** HTTP header in entry point JSPs (the ones that directly respond to HTTP requests rather than being included by other JSPs).

9.8.3 User Registration

If you expect large numbers of users (say 10 000) to register on your site within a very short space of time (say 5-10 minutes), then you will need to establish some kind of queueing mechanism to cope with this.

9.9 How to Test

In order to know whether or not your installation is likely to meet your needs you need to test it. The following sections provide some advice on testing and useful test tools. Three kinds of testing are considered:

- Smoke testing (initial tests intended to give you a general idea of how your set-up is performing)
- Functional testing (does your set-up actually do all the things it's supposed to do?)
- Load testing (will your set-up function satisfactorily under the maximum loads you expect your site to experience?)

9.9.1 Smoke Testing

A good starting point is to verify that the site is actually delivering content and to measure how fast it does this over time. You can do this by repeatedly accessing the site using the **wget** command and

- Observing the effect on operating system resources using commands such as **top**, **vmstat** and **iotop**
- Observing how the Content Engine responds using the performance summary pages in the **escenic-admin** web application (see section 2.1.4)

wget downloads a requested page with all its linked resources, such as images, style sheets and Javascript files. You should always call it several times when you are testing, in order to even out variations in performance. The time taken to respond to a single request cannot be trusted, since it may have arrived at an exceptionally good or bad point in time: when the caches are being filled up, when the connection to the database needs to be re-established or when Java is performing garbage collection. You should therefore submit the command in a loop that executes it a number of times, for example:

```
$ for i in $(seq 10); do
    time \
    wget -p \
    --delete-after \
    -o /dev/null \
    http://mysite.com/
done
```


You should repeat this test at intervals to see the effect of the changes you make during tuning.

This command can also be used to fill up the front end caches after they have been flushed (for instance after a new deployment of your portal software).

9.9.2 Functional testing

We recommend using JMeter for functional tests. You can use it to write scripts that simulate typical user activities. We do not, however, recommend JMeter for load testing. It does not put enough strain on an installation to verify that it can sustain real, high volume traffic.

9.9.3 Load testing

For load testing we recommend two different tools:

- Siege for testing straightforward read operations. Siege is multi-threaded and can exert enough pressure on your site to quickly reveal its weaknesses.

  Here is an example **siege** command for starting 100 sessions on an Escenic Community Expansion site, and creating 50 blogs in each session:

  ```
  $ $ siege -c 100 \
    -r 50 \
    -f siegedata-create-blog \
    -H "Cookie:...."
  ```

  The actual HTTP request sent to the browser is read from a siege data file (**siegedata-create-blog** in the example above). These files have a very simple format, for example:

  ```
  http://mysite.com/community/addStory.do POST
   parameterOne=valueOne&parameterTwo=valueTwo...
  ```

  They can easily be constructed by carrying out an operation in the browser and then using a debugger such as Firefox's Firebug to capture what is actually being sent to the server.

- httperf for more testing more complex scenarios involving user input. **httperf** allows you to write session scripts that simulate the GET, PUT, POST and DELETE operations various kinds of user activity would result in. Furthermore, it can replay your Apache access logs, giving your tests real user traffic patterns as opposed to looping through a list of URLs sorted in alphabetical order.

  Here is an example that shows **httperf** creating 1000 connections and submitting 20 requests over each connection, establishing 100 connections per second:

  ```
  $ httperf\
    --hog \
    --server myserver.com \
    --num-conn 1000 \
    --ra 100 \
    --num-calls=20
  ```

  See the **httperf** man pages for a detailed explanation of the parameters.

Once you have built up a library of tests, you can create a shell script to execute them all simultaneously. For example:

```
#! /usr/bin/env bash
create_blog.siege &
commit_poll_vote.siege &
login_user.siege &
```

```
replay_the_access_log.httperf &
```

10 Backup

There are three items that need to be backed up in order to have a full backup of an Escenic installation:

- The database server
- Various files in the file system

10.1 Database Server

All publication content other than images and media files are stored in the database. Database backups should be carried out every day, ideally at a time of day when little new content is created.

For information on how to carry out and verify database backups, see the documentation for your particular database server.

Note that if your database server needs to be shut down during backups, then your publications will be partly inaccessible to users. Partly inaccessible means:

- No updates will be possible
- Any previously accessed pages that have not been removed from the cache will be accessible to readers; others pages will not be accessible.

Most database servers do, however, support online backup.

10.2 File System

The following kinds of file system files need to be backed up:

- Data files
- Content Engine configuration files
- Publication web applications
- Content Engine program files

There are many utilities available, both commercial and open source, for carrying out file system backups. You can either use one of these or write your own backup script.

10.2.1 Data Files

The data files that need to be backed up consist of publication images and media files, which are not stored in the database. The location of these files is defined by the **`ServerConfig`** component's **`filePublicationRoot`** property. Use the **`escenic-admin`** application's **Component browser** option (see section 2.1.15) to see this property.

All this folder's sub-folders and files should be backed up. Backups should be performed on the same schedule as the database, since the files stored here are closely related to database content.

10.2.2 Content Engine Configuration Files

Depending on your configuration set-up you may have one or more configuration layer on each server that needs to be backed up. For further information about configuration layers and their locations, see chapter 4.

You are strongly recommend to keep all your configuration layers in some kind of version control system, so that you can easily track what changes have been made and revert to earlier versions if the system should become unstable after configuration changes. If you do this, then you will not need to keep backups of these files (but you will, of course need to keep backups of your version control system repository).

Backups should be performed daily.

10.2.3 Publication Web Applications

The web applications that drive Escenic publications consist of a combination of template code (JSP files) and various configuration files in the **`WEB-INF`** and **`META-INF`** folders, which also need to be backed up. These applications are deployed on the application server by the Content Engine assembly tool from a copy in the **`/opt/escenic/assemblytool/publications`** folder.

As with the Content Engine configuration files, you are strongly recommend to keep your publication web applications in a version control system. If you do this, then you will not need to keep backups of the deployed web applications, but you will need to keep backups of your version control system repository.

10.2.4 A Simple Backup Script

Here is a very simple script that saves back up copies of a MySQL database:

```
#! /bin/bash

dir=/var/backups/escenic

# db backup
mysqldump ecedb | gzip -9 > $dir/$(date --iso)-ecedb.sql.gz
```

If you save it in **`/etc/cron.daily/ece`**, then it will be run every day, creating daily backups of your databases.

11 Logging

The Content Engine uses the Apache **`log4j`** utility to handle logging. **`log4j`** is very flexible: among other things, it allows the logging level to be changed without restarting the Content Engine.

By default, the Content Engine outputs log messages to **`System.out`**, which means the application server's log file. You can, however, change this (and many other log settings) by creating a **`trace.properties`** file and adding it to your application server's classpath. An easy way of doing this is:

1. Copy the supplied template **`trace.properties`** file from **`/opt/escenic/engine/classes`** to the root folder of your common configuration layer (**`/etc/escenic/engine/common`**).
2. Edit the copied file (see section 11.1).
3. Most application servers have a folder whose contents are automatically added to the classpath. Create a symbolic link to your **`trace.properties`** file in this folder. If you use Tomcat, for example, you can make sure your **`trace.properties`** is added to the classpath by entering:

```
$ cd /opt/tomcat/lib/
$ ln -s /etc/escenic/engine/common/trace.properties
```

If you do this, then any changes you make to **`trace.properties`** will take effect the next time you start the application server.

11.1 Editing trace.properties

You can use the **`trace.properties`** file to configure all aspects of logging, including the following:

- Log file location
- Log file rotation
- Log file layout
- Logging levels
- Multiple log file generation

The following sections contain some hints on how to use **`trace.properties`** to achieve certain objectives, but no more than that. For a full description of all the possibilities offered by **`log4j`** and the **`trace.properties`** file format (which is complicated), see http://logging.apache.org/log4j/1.2/manual.html and http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html.

11.2 Log File Rotation

An application server can generate large numbers of log messages, so if no action is taken, log files can grow unmanageably large. **Log file rotation** solves this problem by starting a new log file at fixed intervals. You can either use a third party log rotation program or else set up **`trace.properties`** so that the Content Engine starts logging to a new file periodically. You can define the period between log files either by time (every 24 hours, for example) or by data volume (every *x* kilobytes).

You can, for example, change the log file once a day by replacing this line in the default **trace.properties**:

```
log4j.appender.FILE=org.apache.log4j.FileAppender
```

with this:

```
log4j.appender.FILE=org.apache.log4j.DailyRollingFileAppender
log4j.appender.FILE.DatePattern='.'yyyy-MM-dd
```

11.3 Logging Level

Logging level determines how many messages the Content Engine outputs to the log file. For general information about this, see section 2.1.11.

Logging level can be set in three different places:

1. In the **trace.properties** file. General, permanent logging level settings should be made here.
2. In the configuration layers. You can set special logging level settings for a particular component in that component's **.properties** file. Any settings made here will override the general settings in **trace.properties** and are permanent. For general information about configuration layers, see chapter 4.
3. Using the **escenic-admin** application's **View the logging levels** option (see section 2.1.11). Any settings made here will override settings made in **trace.properties** and settings made in the configuration layers. The settings are, however, only temporary: they will disappear when the Content Engine is restarted.

In a production environment you are recommended to set the general logging level to **ERROR**.

11.4 Example Logging Set-up

You can use the following example **trace.properties** file as a basis for your own logging configuration. Replace *mycompany* and *MYCOMPANYLOG* with suitable names of your own.

```
log4j.rootCategory=ERROR
log4j.category.com.escenic=ERROR, ECELOG
log4j.category.neo=ERROR, ECELOG
log4j.category.com.mycompany=ERROR, MYCOMPANYLOG

log4j.appender.ECELOG=org.apache.log4j.DailyRollingFileAppender
log4j.appender.ECELOG.File=/var/log/escenic/ece-messages.log
log4j.appender.ECELOG.layout=org.apache.log4j.PatternLayout
log4j.appender.ECELOG.layout.ConversionPattern=%d %5p [%t] %x (%c) %m%n

log4j.appender.MYCOMPANYLOG=org.apache.log4j.DailyRollingFileAppender
log4j.appender.MYCOMPANYLOG.File=/var/log/escenic/mycompany-messages.log
log4j.appender.MYCOMPANYLOG.layout=org.apache.log4j.PatternLayout
log4j.appender.MYCOMPANYLOG.layout.ConversionPattern=%d %5p [%t] %x (%c) %m%n
```

11.5 Changing the Name of trace.properties

If you want to, you can change the name of the logging configuration file by specifying the system property **`log4j.configuration`**. If you specify the property:

```
log4j.configuration=myserver-log4j.properties
```

then the Content Engine will look for its logging configuration in a file called **`myserver-log4j.properties`**. This can be a useful means of changing the logging configuration for different contexts (development, test, production, for example).

11.6 Content Studio Thread Dumps

Occasionally, a situation can arise which causes Content Studio to freeze. If this situation arises, Content Studio will generate a thread dump as a diagnostic aid. If your users experience this kind of problem with Content Studio, you will probably be asked to submit one of these thread dump files to Escenic support.

By default, Content Studio checks the event dispatch thread every 15 seconds. If the event dispatch thread is busy twice in succession, then a thread dump file called **`Escenic-Content-Studio-thread-dump.log`** is generated. It is written to the location defined by the **`java.io.tmpdir`** system property (as defined on the machine where Content Studio is running).

You can change interval between checks by setting the **`property.com.escenic.studio.thread.dump.interval`** property in the *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`** configuration file. Specify the required interval in seconds (as an integer).

You can disable this process by setting **`property.com.escenic.studio.thread.dump.interval`** to **`-1`**.

12 Monitoring

The **escenic-admin** web application contains a number of functions for monitoring various aspects of Content Engine performance:

- **Performance summary** (see section 2.1.4)
- **Top** (see section 2.1.9)
- **View JSP statistics** (see section 2.1.12)

The Content Engine also provides support for monitoring via the **Java Monitoring and Management Console**.

In addition, all the Content Engine's cache components are now JMX-enabled (JMX stands for **Java Management Extensions**). This means that you can use any JMX client (such as **jconsole**, the Java Monitoring and Management Console bundled with the Java runtime) to monitor cache activity.

If you use the **/usr/local/bin/ece** script to start and stop the Content Engine (recommended, see Install the ece Script), then you can enable and configure JMX support by setting the following parameters in **/etc/escenic/engine/ece.conf**:

enable_remote_monitoring
Set to 1 to enable JMX.

remote_monitoring_port
Specify the number of the port you want to use for monitoring the Content Engine.

If you do not use the **/usr/local/bin/ece** script, then you should set the corresponding Java system parameters (**com.sun.management.jmxremote** and **com.sun.management.jmxremote.port**). On Java 6 JMX is enabled by default, so **com.sun.management.jmxremote** can be omitted.

When you run the JMX client you will need to enter the name or IP address of the host on which the Content Engine is running and the number of the remote monitoring port you are using. If you use **jconsole**, then you will find two Content Engine-related namespaces on the **MBeans** tab:

com.escenic.cache
This contains attributes and statistics for all Content Engine caches.

com.escenic.jvm
This contains Content Engine-related JVM statistics.

13 System Properties

The Content Engine will use the system properties described below if they are specified.

The general method of setting system properties depends on which application server you use. Some application servers allow you to set them as **-D** options in the application server startup command, some read configuration files, some let you set system properties from an administration user interface. Consult the documentation for your application server to find out the best way to set system properties.

> Some system properties are set by the Content Engine's **ece** start-up script, so if you use this script to start the Content Engine, then you can also modify the settings of these properties by editing **/etc/escenic/engine/ece.conf**. You should avoid setting system properties in both places, since which setting will take precedence in such cases is application server-dependent.
>
> The following descriptions indicate which system properties are set by the **ece** start-up script.

There are sensible defaults for all system properties, so they do not necessarily need to be explicitly set.

java.security.policy
: Overrides the default java security configuration. Value: **[some location of your choice]/java.policy**. The file **java.policy** should be copied to the file system of the application server from **ECE_CONFIG/security/**

java.security.auth.login.config
: Overrides the default java security configuration. Value: **[some location of your choice]/jaas.config**. The file **jaas.config** should be copied to the file system of the application server from **ECE_CONFIG/security/**

  For WebLogic installations, use **[some location of your choice]/jaas-weblogic.config**. The file **jaas-weblogic.config** should be copied to the file system of the application server from **ECE_CONFIG/security/**

com.escenic.instance
: The property **com.escenic.instance** will automatically have the value of the name of the host that the instance runs on if both **escenic.server** and **com.escenic.instance** are left unspecified. Set this property if you want it to a have a different value than the host name. One scenario that requires this property to be set is when you are running two application server instances on the same host. Its value should only consist of only letters, numbers, dots and hyphens.

  The property **com.escenic.instance** used to be the **escenic.server** property. The property **escenic.server** still works but it is deprecated. Content Engine will ensure that **escenic.server** and **com.escenic.instance** have the same value. If both are set by your configuration, Content Engine will ignore **escenic.server** and assign your value of **com.escenic.instance** to the **escenic.server** property.

com.escenic.instance.class
: The property **com.escenic.instance.class** defaults to the basename of the the EAR file at assembly time. Usually, this is "engine" since the EAR file name is **engine.ear**. The name is taken from the server class of the EAR file.

If you copy the **`default.properties`** (which describes the default **`engine.ear`**) and have more than one server class it will be possible to use the name of your server class in your configuration files using the **`${com.escenic.server.class}`** syntax.

Assembly tool will create one ear file for every property file that it finds in the **`serverclasses`** directory. Each of these will run with **`com.escenic.instance.class`** property set to the ear file name.

Only set up this property if you want it to a have a different value than the ear filename (the server class name). It should only consist of letters, numbers, dots and hyphens.

14 Content Studio Setup

This chapter contains information about various issues related to the set-up of Content Studio.

14.1 Language and Country Settings

By default, Content Studio uses the language and country settings of the client machine's operating system if possible. If this is not possible (because the required language files are not available), then it uses English language and country settings by default.

You can force Content Studio to ignore the operating system settings and use a specified language by adding the following properties to *configuration-layer-root*`/com/escenic/webstart/StudioConfig.properties`:

`property.language`
The 2-letter ISO language code of the required language (for example, **`en`**, **`no`** or **`de`**). This setting determines the language in which all Content Studio menu items, labels and messages are displayed. If you specify a language code for which no text files are available, then English will be used by default.

`property.country`
The 2-letter ISO country code of the required country (for example, **`US`**, **`NO`** or **`DE`**). This setting determines the formats and conventions used for displaying such things as decimal numbers, dates and so on.

In addition, it is possible for Content Studio users to override the default behavior **and** any settings in **`StudioConfig.properties`** by supplying parameters in the Content Studio URL. The default Content Studio start-up link has the following URL:

```
http://host:8080/studio/studio.jnlp
```

where *host* is the host name or IP address of the Content Engine host. A user can force Content Studio to be run in German, for example, by entering the following URL in the browser address field instead of just clicking on the start-up link:

```
http://host:8080/studio/studio.jnlp?language=de&country=DE
```

Text files for the following languages are currently supplied with Content Studio:

- English
- German

14.1.1 Translating Content Studio

If you need to make Content Studio available in a language other than those supplied by Escenic, you can translate it yourself. For information on how to do this, see the Content Studio Translation Guide.

14.2 Spelling Dictionaries

Spelling dictionaries for the following languages are supplied with Escenic Content Studio:

- English
- German
- Spanish
- French

You can however, add dictionaries for other languages (including right-to-left languages such as Arabic).

Content Studio uses a third-party spelling checker created for an XML editor called XmlMind. This product requires dictionaries to be compiled into a proprietary format, so in order to create a dictionary for Content Studio you must first download a (free) dictionary compiler from `http://www.xmlmind.com/xmleditor/dictbuilder.shtml`.

`dictbuilder` is a Java program. Supplied with it are a shell script and .BAT file so that it can be used as a command line utility on any standard operating system. Full documentation is also available at the above location.

Once you have downloaded and installed **`dictbuilder`**, the basic procedure for adding a dictionary to Content Studio is:

1. Obtain one or more plain text word lists from which a dictionary can be generated. If you use more than one word list, they must be in the same encoding.
2. Obtain or create a **hints file**: this is a text file containing optimization rules for the target language. Ready-made hints files are provided for a number of languages in the **`dictbuilder`** download package. If there is no hints file in the package for your target language, then you will need to create one. In order to create a good hints file you need detailed knowledge of the target language. For further information, see `http://www.xmlmind.com/xmleditor/_dictbuilder/doc/hints_file.html`.
3. Optionally, obtain or create a **freq file** (a list of frequently-used words) and a **prefixes file** (a list of allowed prefixes). Again, these are provided for some languages in the **`dictbuilder`** download package, otherwise you can make them yourself if you have sufficient knowledge of the target language.
4. Generate a dictionary using **`dictbuilder`**. For further information, see `http://www.xmlmind.com/xmleditor/_dictbuilder/doc/using_builder.html`. This produces a **`.cdi`** output file.
5. Optionally package the **`.cdi`** file in a **`.dar`** archive. For further information, see `http://www.xmlmind.com/xmleditor/_dictbuilder/doc/storage_of_dicts.html`.
6. Upload the **`.cdi`** file or **`.dar`** archive to a web server somewhere in your network.
7. Create a text file and enter the URL of the dictionary (**`.cdi`** file or **`.dar`** archive) you have uploaded. If you have created several dictionaries, then you should add the URLs of all your dictionaries to this file, each on a separate line.
8. Upload this text file to a web server somewhere in your network.

9. Edit your *configuration-layer-root*`/com/escenic/webstart/StudioConfig.properties` file. Add a new property called `property.com.escenic.xmlmind.dictionary.list.url`, and set its value to the URL of the dictionary list file that you uploaded in step 7.

Once this has been done, the new dictionaries you have added should be available from any Content Studio instances launched via WebStart.

If , for example, you created dictionaries for Norwegian and Swedish (`no.cdi` and `se.cdi`) and uploaded them to `http://www.my-domain.com/static/dictionaries`, then you would need to create a file (lets call it `dictionary-list.txt`) with the following content:

```
http://www.my-domain.com/static/dictionaries/no.cdi
http://www.my-domain.com/static/dictionaries/se.cdi
```

If you uploaded this file to the same location, then you would need to add the following property to your *configuration-layer-root*`/com/escenic/webstart/StudioConfig.properties` file:

```
property.com.escenic.xmlmind.dictionary.list.url=http://www.my-domain.com/static/
dictionaries/dictionary-list.txt
```

14.2.1 Dictionary Sources

There are a number of free/open source spelling checkers available, many of which have associated word lists for a wide range of languages. These word lists are in most cases themselves open source, and can therefore be freely used (although there may be restrictions on redistribution). They may need to be decompiled from their native format before they can be used as input to `dictbuilder`.

Two of the most commonly used spelling checkers are `ispell` and `aspell`. Dictionaries created for these two systems cover a wide range of languages.

In addition, ready-converted XmlMind dictionaries for a number of languages can be downloaded from http://www.xmlmind.com/xmleditor/spellchecker/user_contrib_dicts.html.

14.2.1.1 Converting Ispell Dictionaries

The `dictbuilder` documentation includes instructions on how to use dictionaries made for `ispell` (a popular open source spelling checker) here: http://www.xmlmind.com/xmleditor/_dictbuilder/doc/from_ispell.html.

14.2.1.2 Converting Aspell Dictionaries

Here is an example of how to create an XmlMind dictionary from an `aspell` dictionary (in this case Greek, which has the ISO language code `el`). To do this you need to install `aspell` on your computer as well as `dictbuilder`. `aspell` is available for both Windows and Unix-based platforms. The `dictbuilder` package includes a hints and a freq file for Greek.

1. Export the `aspell` word list:

   ```
   $ aspell --encoding ISO-8859-7 -l el dump master > greek.txt
   ```

   Note that you need to ensure that the encoding of the exported word list is the same as any hints, freq and prefixes files you are going to use.
2. Convert the exported word list with `dictbuilder`:

   ```
   $ dictbuilder -cs ISO-8859-7 -hints el.hints -freq el.freq greek.txt -o el.cdi
   ```

3. Upload **`el.cdi`** to your web server.
4. Create a **`dictionary-list.txt`** file containing the URI of **`el.cdi`** (plus the URIs of any other dictionaries you have uploaded). For example:

```
...
http://www.my-domain.com/static/dictionaries/el.cdi
...
```

5. Upload **`dictionary-list.txt`** to the same location.
6. Add the following property to your *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`** file:

```
property.com.escenic.xmlmind.dictionary.list.url=http://www.my-domain.com/static/
dictionaries/dictionary-list.txt
```

14.3 Memory Settings

You can change the Java memory settings used to run Content Studio on the client by modifying the **`vmargs`** property in *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`**. The default setting for this property is:

```
vmargs=-Xms256m -Xmx1024m
```

These settings may, however, be too low for users who need to be able to edit large images.

14.3.1 Size Restrictions in Image Editor

Content Studio has a threshold which, if crossed, disables image editing features. This is to avoid using too much memory. Opening images in external editors is still supported in this mode.

By default, the threshold is set to 2 megapixel images. This can be changed by setting the property **`com.escenic.studio.maximum-editable-image-size`** in *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`** as follows:

```
property.com.escenic.studio.maximum-editable-image-size=size
```

where *size* is the number of pixels that designate the threshold, above which Content Studio will disable editing functionality.

14.4 Default Sort Order

You can change the default sort order Content Studio uses when listing content items by setting the property **`com.escenic.studio.search.date`** in *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`** as follows:

```
property.com.escenic.studio.search.date=date
```

where *date* is one of:

***creationDate* (default)**
: Content items are sorted by the date they were created.

lastModifiedDate
Content items are sorted by the date they were last modified.

publishDate
Content items are sorted by the date they were published. Note that if you use this option, content items that have never been published will not be listed.

14.5 MIME Type Mappings

You can override Content Studio's default MIME type mappings by setting the property **`com.escenic.external-mimetypes`** in *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`** as follows:

```
property.com.escenic.external-mimetypes=mimetypes
```

where *mimetypes* is a JSON object defining a series of mappings between file type extensions and MIME types. For example:

```
property.com.escenic.external-mimetypes={avi:'video/x-msvideo',jpg:'image/jpeg'}
```

Content Studio's default MIME type mappings usually work for most common cases. It is only necessary to use this property if you get problems that arise from incorrect MIME type assignment.

14.6 Clipboard Size

The Content Studio clipboard panel can hold up to 200 content items by default. You can change this limit by adding the following properties to *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`**:

`property.com.escenic.studio.clipboard.size`
The maximum number of content items to be displayed in the clipboard panel. The default value is 200.

`property.com.escenic.studio.clipboard.warning.size`
A limit above which a warning icon is displayed, indicating that the clipboard panel will soon be full. The default value is 180.

Reducing the size of these properties lowers Content Studio's memory usage, so it may be a good idea if users are experiencing memory-related performance problems.

14.7 Varnish and Content Studio File Uploads

Content Studio and Tomcat use a chunked transfer encoding mechanism to support upload of large files. Varnish (a web application accelerator commonly used on Escenic sites) does not support chunked transfer encoding. This means that if you use Varnish between Content Studio and Tomcat, uploading of files is not possible.

For this reason you are recommended **not** to use Varnish between Content Studio and Tomcat.

If not using Varnish between Content Studio and Tomcat is difficult, you can alternatively disable Content Studio's use of chunked transfer encoding. To do this, add the property **`property.com.escenic.client.chunked`** to *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`** and set it to **`false`** as follows:

```
property.com.escenic.client.chunked=false
```

This instructs Content Studio to not use chunked transfer encoding when posting files.

14.8 Configuring the Content Studio Launcher

Content Studio start-up can be controlled either by **Java Webstart** or by **Getdown**. Java Webstart is the traditional tool for launching desktop Java applications. Getdown is a newer, open source alternative that provides some additional functionality not provided by Webstart. In particular, it allows you to exercise more control over the Java version used to run Content Studio than Webstart does.

Users can choose between Webstart (the default launcher) and Getdown by clicking on the **Alternative login** link on the Content Studio start page (see http://docs.escenic.com/ece-content-studio-guide/5.7/alternative_start_up_procedures.html).

> Users should be encouraged to use Getdown rather than Webstart, since this will give you full control over the Java version used to run Content Studio. For information about how to make Getdown the default choice, see section 14.8.2.

You can configure Getdown to limit the Java versions that Content Studio will run on. You can also configure it to provide a private JVM that will be downloaded and installed if a suitable version is not available on the client machine.

14.8.1 Configuring Getdown

To configure Getdown, you need to set some properties in the **Content Studio webapp configuration layer** (not the same as the Content Studio configuration layer described in section 14.9). This layer is usually located in **`/etc/escenic/engine/webapp/studio/`**. You need to set the following properties in *configuration-layer-root*/**`com/escenic/webstart/ApplicationGetdownFactory.properties`**:

`getdowntxt.java_min_version`

The oldest Java version on which Content Studio will run. If the client does not have this version or a more recent version available then either:

- If you have specified a download version for this client with **`getdowntxt.java_location`**, then that version will be downloaded and used as a private JVM for Content Studio.
- If there is no download version available for this client, then Content Studio will not be started.

The version number is specified as a single integer constructed from the Java version number *MAJ.MIN.REV_PATCH* using the following formula:

PATCH + 100 * (*REV* + 100 * (*MIN* + 100 * *MAJ*))

The Java version 1.8.0_25, for example, is specified as the number 1080025.

getdowntxt.java_max_version
The newest Java version on which Content Studio will run. You can optionally specify this in addition to **getdowntxt.java_min_version**. The version number must be calculated using the same formula. If you specify this property then only Java versions in the specified range will be used, and if there is no such Java version on the client, then a download version will be used if available, and otherwise Content Studio will not be started.

getdowntxt.java_exact_version_required
If this property is set to **true**, then only the exact Java version specified with **getdowntxt.java_min_version** will be used. If specified, **getdowntxt.java_max_version** is ignored. If the version specified with **getdowntxt.java_min_version** is not available on the client, then a download version will be used if available, and otherwise Content Studio will not be started.

getdowntxt.java_location
This property is used to hold the download URLs of JVM packages suitable for use on various clients. Any number of different URLs can be specified in the following comma-separated format:

```
[platform] url, [platform] url...
```

where

- *platform* is the name of the client OS as returned by the Java method **System.getProperty("os.name")** (for example, **windows 7**, **windows 8** or **mac os x**)
- *url* is the URL of the JVM package to be downloaded. It can either be a local absolute URL (starting with **/**) if the package is stored on the local host, or a fully-qualified URL starting with **http:** or **https:** if the package is stored on a different host. For more information about how to obtain or create suitable Java packages, see section 14.8.1.1.

If you want to be able to support both 32- and 64-bit Windows machines, you can replace the **[***platform***]** identifier with a composite identifier of the form **[***platform-architecture***]** where *architecture* is the name of a machine architecture as returned by the Java method **System.getProperty("os.arch")** (for example, **i386**, **x86_64**, **amd_64**).

The following example shows a set-up that enforces use of Java 1.8.0_25 only. It provides downloads of this version for Mac and Windows clients (32-bit and 64-bit):

```
getdowntxt.java_min_version = 1080025
getdowntxt.java_exact_version_required = true
getdowntxt.java_location = [windows-i386] /studio/client/java_windows_i386.jar,
 [windows 7-x86_64] /studio/client/java_windows_x86_64.jar,[mac os x] /studio/client/
java_osx.jar
```

14.8.1.1 Getdown Java Packages for Content Studio

Getdown requires the JVMs it installs to be packaged into JAR files that have the same structure as an installed version of that JVM on the target platform. In addition, Content Studio requires the use of Oracle Java only, version 1.7 or 1.8. To create such a package:

1. Install the required version of Oracle Java on a target client machine.
2. Test the installation to make sure it works.

3. Find the JRE root folder. Note that Java installation trees vary between platforms, versions, and according to whether or not you have installed just the Java runtime or the Java Development Kit. The JRE root folder is the parent of the **bin** folder that contains the Java executable:

   jre-root/**bin/java.exe** on Windows
   jre-root/**bin/java** on Macs
4. When you have found the JRE root folder, copy it to a new location.
5. Rename your copied folder to **java_vm**.
6. Create a ZIP archive of the **java_vm** folder (**java_vm.zip**). The ZIP archive should contain a single folder called **java_vm** (not just the folder's contents).
7. Rename the ZIP file, giving it a **.jar** extension (**java_vm.jar**).
8. Delete your copied **java_vm** folder.

14.8.1.2 Upgrading Getdown Java Packages

Getdown does not have any built-in mechanism for installing upgraded downloaded packages. If you upgrade your downloadable Java packages to a new version, then by default Getdown will only download the upgraded Java versions to new client machines where it has not already downloaded a private version of the JRE. In order to force an upgrade on clients where Getdown has already downloaded a private version of the JRE you must manually delete the old JRE installation so that a new download is required. You will find the folder here:

C:\Users*username***\com.escenic.studio***engine-host***\java_vm** (Windows)
/Users/*username***/com.escenic.studio/***engine-host***/java_vm** (Mac)

where *engine-host* is the domain name of your Content Engine host.

14.8.2 Making Getdown the Default Launcher

You can make Getdown the default launcher for Content Studio by adding the following property setting to *configuration-layer-root*/**com/escenic/webstart/StudioConfig.properties**:

```
defaultLauncher=getdown
```

14.9 Configuring Content Studio Preview

14.9.1 Configuring Preview Devices

Content Studio's content editors and section page editors provide preview functionality that allows the user to see what the content item or section page they are working on will look like on various devices. They are able to switch between 3 main previews corresponding to 3 device types: PCs/laptops (large), iPads/tablets (medium) and mobile phones (small). Each of these device categories can, however, include any number of specific devices that the user can select from drop-down lists.

You can configure what entries should appear in each of these lists: the names of the devices, and their screen resolution. To do so you need to create an XML file that looks something like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<device-list>
```

```
    <large>
        <device>
            <name>Default</name>
        </device>
        <device>
            <name>TV</name>
            <width>1980</width>
            <height>1080</height>
        </device>
    </large>
    <medium>
        <device>
            <name>iPad</name>
            <width>768</width>
            <height>1024</height>
        </device>
        <device>
            <name>Galaxy III</name>
            <width>768</width>
            <height>1024</height>
        </device>
    </medium>
    <small>
        <device>
            <name>iPhone 5</name>
            <width>640</width>
            <height>960</height>
        </device>
    </small>
</device-list>
```

> You can omit the **`width`** and/or **`height`** elements from any device configuration if you wish. If you do this then the preview will use the maximum space available. The **`Default`** device in the above example has no **`width`** or **`height`** elements, with the effect that the **`Default`** preview displays as it would in an ordinary browser window.

You then need to add an entry that references this file to *configuration-root*/**`com/escenic/studio/preview/DeviceSelection.properties`** in your Content Studio configuration layer:

```
configXML=custom-device-file-url
```

where *custom-device-file-url* is the URL of the file you have created. You can specify either a relative or absolute URL.

For a description of how to create a studio configuration layer, see section 14.10.

A sample custom device file that you can copy and edit is included in the Content Engine distribution:

`/opt/escenic/engine/siteconfig/config-studio-skeleton/com/escenic/studio/preview/device.xml`.

An RNG schema that you can use to validate your finished custom device file is also included in the Content Engine distribution: **`/opt/escenic/engine/schemas/device.rng`**.

> The technique described above will only work for **responsive design** publications like those produced by recent versions of the Widget Framework. In a responsive design publication, pages include CSS media queries and Javascript that automatically adjusts layout and content to suit the

current display device. For older publications that do not use responsive design techniques, and rely on the use of separate sites for desktop and mobile devices, some additional configuration is required. For details, see section 14.9.1.1.

14.9.1.1 Configuring Preview Devices for Traditional Mobile Sites

If your publication is generated in separate versions for desktop and mobile devices, then you need to:

- Add some additional configuration information to the **device.xml** file described in section 14.9.1.
- Add some logic to ensure that the correct templates are used when displaying the preview.

Assuming you have just one desktop site and one mobile site (the usual case), then you can drop one of the top level elements (**medium**, for example). The additional configuration you require consists of **param** elements that you can add to the mobile device definitions:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<device-list>
    <large>
        <device>
            <name>Desktop</name>
            <width>1280</width>
            <height>1024</height>
        </device>
    </large>
    <small>
        <device>
            <name>IPhone 5S</name>
            <width>320</width>
            <height>568</height>
            <params>
              <param name="device" value="iphone5s" />
            </params>
        </device>
        ...
    </small>
</device-list>
```

The **param** elements specified in the file result in the addition of corresponding parameters to the generated preview URLs. For example:

```
http://editorial-server:port/article-id.ece?token=preview-token&device=iphone5s
```

Adding such parameters to one or both device definitions makes it possible to differentiate between mobile and desktop preview URLs. You then need to set up some mechanism to make use of the information and ensure that the correct preview is displayed. Some possible methods include:

- If the publication serves different templates based on the HTTP **User-Agent**, add a filter to the servlet filter chain that that sets the correct **User-Agent** based on the device parameter in the preview URL. The example preview URL shown above, for example, might result in the following **User-Agent** string:

  ```
  User-Agent: "Mozilla/5.0 (iPhone; CPU iPhone OS 5_0 like Mac OS X)
   AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9A334 Safari/7534.48.3"
  ```

- Use Apache or Varnish to redirect requests containing the parameters you have defined to a different server, and to set the appropriate **User-Agent** as above.

> The Content Studio preview interface is fixed and always displays three device options: **Large**, **Medium** and **Small**. Removing the **`medium`** element from your **`device.xml`** file will not remove the **Medium** option from the preview function, but just render it useless. Note also that the orientation buttons in the Content Studio preview interface will not work.

14.9.2 Configuring Multi-Host Preview

Content Studio loads the preview inside an IFrame. If you are running a multi-host installation, then by default the preview user interface is provided by the **editorial-host**, but the actual preview (the content of the IFrame) comes from the **presentation-host**. In this situation, cross-site scripting restrictions will prevent Content Studio from communicating with the preview. To overcome this problem you must do one the following:

- Set the **`publication.previewURL`** parameter in each publication's **`feature`** resource to point to the **editorial-host**. If you do this then both the preview user interface and the preview itself will be loaded from the **editorial-host**. See publication.previewURL
- Make sure your JSP templates include the following code:

  ```
  <c:if test="${requestScope['com.escenic.context.isPreview'] == 'true'}">
    <c:url var="js" value="/js/cross-site-preview.min.js" scope="request"/>
      <script
          type="text/javascript"
          src="${js}"
          data-studio-host="http://editorial-host:port"></script>
  </c:if>
  ```

  The test for **`com.escenic.context.isPreview`** ensures that the script **`cross-site-preview.min.js`** is only included for previews. The attribute **`data-studio-host`** is optional, but you are recommended to use it. Set it to point to your **editorial-host**. This restricts the script to only communicate with the **editorial-host**.

14.9.3 "Old-Style" Preview

You can force Content Studio to revert to an older type of preview functionality by setting the following property in *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`**:

```
property.com.escenic.studio.preview.button=true
```

Instead of displaying a preview tab, Content Studio will then display a **Preview** button that displays a preview in your browser. No device-specific preview functionality is provided.

14.10 Content Studio Configuration Layer

In addition to the Content Engine configuration layers described in section 4.1, there is also a Content Studio configuration layer. By default this configuration layer is located in **`/etc/escenic/engine/studio`**. You can, however, change its location by adding a **`configDirectory`** property to *configuration-layer-root*/**`com/escenic/webstart/StudioConfig.properties`** in one of your Content Engine configuration layers. For example:

```
configDirectory=/path/to/studio/configuration/layer
```

14.10.1 Creating a Studio Configuration Layer

A skeleton configuration layer for studio is supplied in **`/opt/escenic/engine/siteconfig/config-studio-skeleton`**. To create a configuration layer from this skeleton, log in as **`escenic`** and copy the configuration layer to **`/etc/escenic/engine/studio`**.

```
$ cp -r /opt/escenic/engine/siteconfig/config-studio-skeleton/* /etc/escenic/engine/
studio/
```

You can now configure your Content Studio installation by modifying the **`.properties`** files in the **`/etc/escenic/engine/studio/`** tree.

14.11 Configuring The Performance Monitor

The Content Studio status bar contains a performance monitor. It monitors the performance of Content Studio and lights up when performance problems arise. You can determine at what point the monitor will light up for various performance factors by editing the **`.properties`** files in the *configuration-root***`/com/escenic/studio/performance`** folder of your Content Studio configuration layer.

For a description of how to create a studio configuration layer, see section 14.10.

14.12 Configuring Content Studio Search Results

By default, the search function in Content Studio returns a maximum of 200 results. You can, however, change this default by editing **`DefaultSearchPlugin.properties`** in the *configuration-root***`/com/escenic/studio/search`** folder of your Content Studio configuration layer. Open the file for editing and set **`maxSearchLimit`** to the required maximum limit. For example:

```
maxSearchLimit=100
```

Setting this limit to a very high value will affect performance.

For a description of how to create a studio configuration layer, see section 14.10.

14.13 Customizing Content Studio Fonts and Layout

You can modify the appearance of Content Studio in the following ways:

- Set a default font for the whole window
- Override the default font for the text content of specific content item fields
- Apply full-scale CSS formatting to the content of specific rich text fields

14.13.1 Setting the Content Studio Default Font

You can set a default font for Content Studio by adding font properties to *configuration-layer-root***`/com/escenic/webstart/StudioConfig.properties`** in one of your Content Engine configuration layers. You can add multiple properties to specify different default fonts for different operating systems. Each property must have the following form:

```
property.com.escenic.studio.font.windowsvista=Arial
property.com.escenic.studio.font.macosx=Geneva
```

where:

- *os-identifier* identifies an operating system. OS identifier names are formed from the strings returned by the Java **`System.getProperty("os.name")`** method. The string returned by this method is converted to lower case and then all spaces and non-alphanumeric characters are removed. In this way the Java OS name **`Windows 8`** is converted to the identifier **`windows8`**.
- *font-family* is a font family name such as **`Arial`** or **`"Times Roman"`** rather than the name of a single font such as **`"Arial Bold"`**. If the font family name includes spaces, then you must enclose it in quotation marks.

The following settings, for example, specify default fonts for both Windows 8 and Mac OS X:

```
property.com.escenic.studio.font.windows8=Arial
property.com.escenic.studio.font.macosx=Geneva
```

Setting a default font changes all the text appearing in the Content Studio window, including menus, button labels and so on, as well as user-defined content.

14.13.2 Overriding the Default Font

You can modify the appearance of the text in specific content item fields by adding **`ui:style`** elements to the field definitions in a publication's **`content-type`** resource. For example:

```
<field mime-type="text/plain" type="basic" name="title">
  <ui:style>body { font-size: 20px; font-weight: bold; }</ui:style>
</field>
```

You can use this technique to change the appearance of any content item field that displays text or numbers: basic, number, schedule, date, URI and so on.

The **`ui:style`** element accepts CSS content, but for any field type other than a rich text field the CSS that is recognized is limited as follows:

- You must use the selector **`body`**. A CSS rule with any other selector will be ignored
- The only properties that are used are **`font-size`** and **`font-weight`**
- **`font-size`** must be specified using either pixel (**`px`**) or percentage (`%`) units. Any other units will be ignored
- The only **`font-weight`** values that are recognized are **`normal`** and **`bold`**

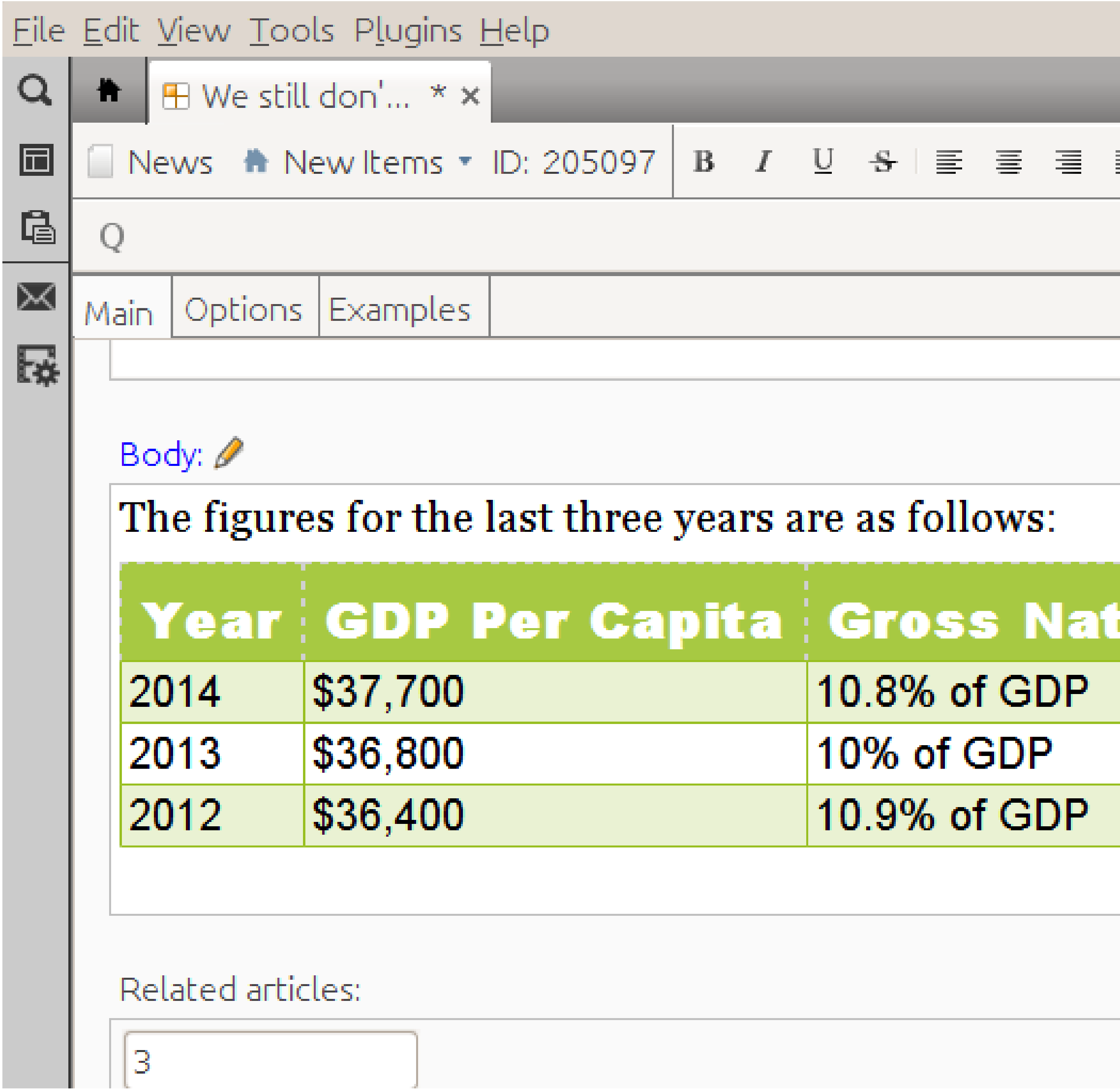
14.13.3 Formatting Rich Text Fields

You can modify the appearance of rich text fields (basic fields with the MIME type **`application/xhmtl+xml`**) using **`ui:style`** elements in the same way as other fields. In this case, however, since the field contains XHTML, you can make full use of CSS to control the layout and appearance of the field content. You can, for example, supply a full-scale CSS file that to some extent mimics the appearance of the published content.

The following example shows a body field definition with CSS defining a nice table layout:

```
<field name="body" mime-type="application/xhtml+xml" type="basic">
  <ui:label>Body</ui:label>
  <ui:layout rows="10" cols="40"/>
  <ui:style>
    body {
        font-size: 20px;
        font-family: Georgia, Times-Roman, serif;
        width: 100%;
        border-collapse: collapse;
    }
    table {
        font-family: Arial, Helvetica, sans-serif;
    }
    td {
        text-align: left;
        border: 1px solid #98bf21;
    }
    th {
        font-family: "Arial Black";
        font-size: 24px;
        text-align: left;
        padding: 10px 10px 10px 10px;
        background-color: #A7C942;
        color: #ffffff;
    }
    tr:nth-child(even) td {
        color: #000000;
        background-color: #EAF2D3;
    }
  </ui:style>
</field>
```

The result looks like this:



For further information and additional examples, see http://docs.escenic.com/ece-resource-ref/5.7/ih_style.html.

15 Third Party Authentication

The Content Engine can be set up to use a third party for authentication of users, instead of doing the user authentication itself. Three third party authenticators are supported:

- Microsoft Active Directory
- Google Apps
- Facebook

This means that users in organizations with primarily Windows-based networks and users in organizations that use Google Apps as their standard office suite can log in to Content Studio and Web Studio using their "ordinary" user names and passwords. It is also possible to allow the use of Facebook IDs for authentication where appropriate. Note, however, that:

- This is more of a "federated login" mechanism than "single sign on": users will still have to log in when starting Content Studio, Web Studio or Mobile Studio, even if they are already logged in to Active Directory/Google Apps/Facebook.
- Only authentication is carried out by the third party, authorization is still performed by the Content Engine, so you still have to define Content Engine users. The Content Engine users must have identical user names to the Active Directory/Google Apps/Facebook users.

The general procedure for setting up third party authentication is:

1. Using Web Studio, create users (see Create New User) for all the existing Active Directory/ Google Apps/Facebook users who are to use Content Studio, Web Studio or Mobile Studio. The user names you specify must be identical to the user names in Active Directory/Google Apps/ Facebook. You must leave the password fields blank.

   > You can also migrate **existing** Escenic users to ActiveDirectory/Google Apps/Facebook by changing their user name in Web Studio to match an existing user name in the third party system.

2. Assign access rights to these user in the usual way (see Editing Users and Persons) .
3. If you have any existing Content Engine users that you want to move over to Active Directory/ Google Apps/Facebook, then you can do so by:
   - Adding users with identical user names to Active Directory/Google Apps/Facebook
   - Removing the password from the user record in Web Studio

   > You do not **have** to move all your Content Engine users to the third party authentication system. Any users that you do not transfer will continue to work as before. (In the case of Active Directory, whether or not this is the case actually depends on your set up - see section 15.1.2.)

4. Set up the Content Engine to use to the third party authenticator. This process is different for each of the supported third-party authenticators, but in both cases it involves reassembling and redeploying the Content Engine For details see either section 15.1, section 15.2 or ??.
5. Using Web Studio, you can now tidy up by deleting any old Content Engine-authenticated users that are no longer required (see Person and User Archive).

15.1 Active Directory Authentication

Carry out the following tasks to set up Active Directory-based authentication.

15.1.1 Enable Connection to Active Directory

In order to enable the use of Active Directory you need to create a configuration file defining how to connect to Active Directory, and deploy it together with the Content Engine as follows:

1. Login as **`escenic`** on your **assembly-host** (see the Escenic Content Engine Installation Guide for an explanation of this term).
2. Go to the location of the assembly tool's **`classes`** folder:

   ```
   $ cd /opt/escenic/assemblytool/classes/
   ```

3. Create a new directory structure:

   ```
   $ mkdir -p com/escenic/jaas
   ```

4. Create a new file named **`shiro.conf`** in the new directory and open it in an editor. Enter the following configuration settings:

   ```
   [main]
   activeDirectoryRealm = org.apache.shiro.realm.activedirectory.ActiveDirectoryRealm
   activeDirectoryRealm.url=ldap://my_server:3268/
   activeDirectoryRealm.searchBase=dc=my,dc=company
   activeDirectoryRealm.systemUsername=my_username
   activeDirectoryRealm.systemPassword=my_password
   ```

   Set the parameters to match your Active Directory set up:

   `activeDirectoryRealm.url`
   : The URL of your Active Directory server.

   `activeDirectoryRealm.searchBase`
   : The base dn of your Active Directory.

   `activeDirectoryRealm.systemUsername`
   : The user name to use when connection to Active Directory.

   `activeDirectoryRealm.systemPassword`
   : The password to use when connecting to Active Directory.

5. Save the file and build a new ear file by entering:

   ```
   $ ece clean assemble
   ```

6. Deploy the ear file to your **engine-host**s.

15.1.2 Switch to Active Directory

To switch to using Active Directory for authentication you need to change a setting in the Content Engine's authentication configuration file. In a standard installation (as described in the Escenic Content Engine Installation Guide), this configuration file will be located in the common configuration layer: **`/etc/escenic/engine/common/security/jaas.config`**.

Open this file for editing and replace:

```
ece-basic {
        com.escenic.auth.jaas.BasicLoginModule required;
};
```

with one of the following two options:

- ```
  ece-basic {
         com.escenic.auth.jaas.ShiroLoginModule required;
  };
  ```

  This setting completely replaces the Content Engine's native authentication mechanism with Active Directory: only users defined in Active Directory will be able to log in.

- ```
  ece-basic {
        com.escenic.auth.jaas.BasicLoginModule Sufficient;
        com.escenic.auth.jaas.ShiroLoginModule Sufficient;
  };
  ```

  This setting allows both the Content Engine's native authentication mechanism and Active Directory to be used: users with passwords defined in Web Studio will be able to log in as well as users defined in Active Directory.

Restart the application server.

Users should now be able to login to Content Studio and Web Studio using their Active Directory user names and passwords. If this does not seem to work, it may be because Active Directory requires the domain name to be specified with user names. For such case you have to either

- Specify the domain name when login, for example, **`username@example.com`**.
- Or, set the domain name to use by default (see section 15.1.2.1).

For the former option to work properly you must have users having usernames of the same format, i.e. **`username@example.com`** in Escenic Content Engine.

15.1.2.1 Setting a Default Domain

Active Directory may require users to include the domain name with their user name when logging in. That is, they made need to enter something like **`myuser@mydomain.com`** instead of just **`myuser`**.

If this is the case you can fix the problem by modifying the entry in **`jaas.config`** to include a default domain name as follows:

```
ece-basic {
       com.escenic.auth.jaas.ShiroLoginModule required domain=mydomain.com;
};
```

The default domain specified here will then be automatically appended if the user does not specify one.

15.2 Google OAuth Authentication

Carry out the following tasks to set up Google OAuth 2.0-based authentication.

15.2.1 Create a Google Project

Before Content Engine can use Google's OAuth 2.0 authentication system for user login, you must set up a project in Google Developers Console to obtain OAuth 2.0 credentials, set a redirect URI, and (optionally) customize the branding information that your users see on the user-consent screen. For more details, see the Google Developers Console Help.

1. Go to the Google Developers Console.
2. Click **CREATE PROJECT**.
3. Enter a name for the project and click **Create**.

15.2.2 Configure OAuth Authentication

1. Login as `escenic` on your **assembly-host** (see the Escenic Content Engine Installation Guide for an explanation of this term).
2. Go to `/etc/escenic/engine/common/com/escenic/auth/oauth2` and open `OAuth2Configuration.properties` in an editor.
3. Go to the Google Developers Console and select the project you created.
4. In the sidebar on the left, select **APIs & auth**.
5. Find **Google+ API** in the displayed list of APIs and make sure that its **Status** is set to **ON**.
6. In the sidebar on the left, select **Credentials**.
7. Click **CREATE NEW CLIENT ID**.
8. Select **Installed application**.
9. Select **Other**.
10. Click **Create client ID**.
11. Copy the value displayed in the **Client ID** field to the `clientId.application` property in `OAuth2Configuration.properties`.
12. Copy the value displayed in the **Client secret** field to the `clientSecret.application` property in `OAuth2Configuration.properties`. You have now configured OAuth-based authentication for Content Studio only. If you want to use OAuth-based authentication for web applications such as Web Studio as well, continue with step 13. Otherwise you can jump straight to step 20.
13. Click **CREATE NEW CLIENT ID**.
14. Select **Web application**.
15. In the **Authorized JavaScript origins** field, enter `http://`*your-server*.
16. In the **Authorized redirect URI** field, enter `http://`*your-server*`/escenic/logon/oauth2.do`.
17. Click **Create client ID**.
18. Copy the value displayed in the **Client ID** field to the `clientId.web` property in `OAuth2Configuration.properties`.
19. Copy the value displayed in the **Client secret** field to the `clientSecret.web` property in `OAuth2Configuration.properties`.
20. Save and close `OAuth2Configuration.properties`.

15.2.3 Deploy Configuration Changes

1. Build a new ear file by entering:

```
$ ece clean assemble
```

2. Deploy the ear file to your **engine-host**s.

Users should now be able to login to Content Studio (and possible Web Studio and other web applications) using their Google Apps user names and passwords.

15.3 Facebook OAuth Authentication

Carry out the following tasks to set up Facebook OAuth 2.0-based authentication.

15.3.1 Create Two Facebook Apps

Before Content Engine can use Facebook's OAuth 2.0 authentication system for user login, you must create one or two apps on the Facebook Developers website to obtain OAuth 2.0 credentials: one app for Content Studio login credentials, and one app for webapp login credentials (Web Studio and Mobile Studio) if required. For more details, see the Login documentation on the Facebook developer's site.

1. Go to the Facebook Developers web site.
2. Select **My Apps** > **Add a New App** > **advanced setup**. ??is that correct??
3. Fill in the displayed form with suitable values for your Content Studio login app and click **Create App ID**.
4. Solve the displayed CAPTCHA puzzle.
5. A dashboard for the app is now displayed. At the top of the dashboard are an App ID and an App Secret, which you will need to use when configuring authentication. Click on **Settings** (in the menu on the left).
6. Enter your email address in the **Contact Email** field and click on **Save Changes**.
7. Click on **Status & Review** (in the menu on the left)
8. Switch the app on by clicking on the switch next to the question **Do you want to make this app and all its live features available to the general public?**. Confirm that you really want to do it, and the switch's **No** should turn to **Yes**.

??Do you need to do anything else??

If you also want to use Facebook authentication for webapps, repeat steps 2 - 8 in order to create a second app. ??do you do the same things, or something different??

15.3.2 Configure OAuth Authentication

1. Login as `escenic` on your **assembly-host** (see the Escenic Content Engine Installation Guide for an explanation of this term).
2. Go to `/etc/escenic/engine/common/com/escenic/auth/oauth2` and open `OAuth2Configuration.properties` in an editor.
3. Make sure the following properties are set exactly as shown:

```
serviceEnabled=true
name=Facebook
profileUri=https://graph.facebook.com/me
tokenUrl=https://graph.facebook.com/oauth/access_token
authorizationUrl=https://graph.facebook.com/oauth/authorize
```

```
scope=email
userNameProperty=email
expiresProperty=expires
```

4. Go to the Facebook Developers web site and select the first app you created.
5. Copy the value displayed in the **App ID** field to the **clientId.application** property in **OAuth2Configuration.properties**.
6. Copy the value displayed in the **App secret** field to the **clientSecret.application** property in **OAuth2Configuration.properties**. You have now configured OAuth-based authentication for Content Studio only. If you want to use OAuth-based authentication for web applications such as Web Studio as well, continue with step 6. Otherwise you can jump straight to step 10.
7. On the Facebook Developers web site, select the second app you created.
8. Copy the value displayed in the **App ID** field to the **clientId.web** (for Web Studio) and **clientId.mobile** (for Mobile Studio) properties in **OAuth2Configuration.properties**.
9. Copy the value displayed in the **App secret** field to the **clientSecret.web** (for Web Studio) and **clientSecret.mobile** (for Mobile Studio) properties in **OAuth2Configuration.properties**.
10. Save and close **OAuth2Configuration.properties**.

15.3.3 Deploy Configuration Changes

1. Build a new ear file by entering:

```
$ ece clean assemble
```

2. Deploy the ear file to your **engine-host**s.

Users should now be able to login to Content Studio (and possible Web Studio and other web applications) using their Facebook user names and passwords.

16 Read-Only Mode

The Content Engine can be configured to use a read-only connection to the database. Read-only mode can provide a useful means of:

- Ensuring that presentation hosts do not make modifications to the database.
- Scaling up the presentation layer by connecting presentation hosts to a read-only slave database.

When a Content Engine is running in read-only mode:

- All write operations to the database will fail.
- None of the Content Engine's mutex services (services designed to run on only one host) will start.
- Content Engine plug-ins that depend on write access to the database, such as the Viz Community Expansion, Forum and Poll, **will not work**.

16.1 Enabling Read-Only Mode

Enabling read-only mode is very simple: you just set up read-only access to the database and then add one property to a configuration file. You will usually need to add the property in the host configuration layer of each host you want to run in read-only mode (or possibly in a family configuration layer for a group of hosts that are all to run in read-only mode). For general information about configuration layers and how they are organized, see section 4.1.

See Escenic Content Engine Installation Guide for an explanation of the term **database-host** used in the following instructions.

To run two hosts called **`ReadOnly1 and ReadOnly2`** in read-only mode:

1. On your **database-host**, create a user with read-only access to the database.
2. Login as **`escenic`** on **`ReadOnly1`**.
3. Open **`/opt/tomcat/conf/context.xml`** in an editor and change **`username`** and **`password`** to reflect the credentials created in step 1.
4. Login as **`escenic`** on **`ReadOnly2`** and make the same change to **`/opt/tomcat/conf/context.xml`**.
5. Open **`/etc/escenic/engine/host/ReadOnly2/ServerConfig.properties`** in an editor and add

   ```
   readOnly=true
   ```
6. Make the same change to **`/etc/escenic/engine/host/ReadOnly1/ServerConfig.properties`**. (In a standard Escenic installation, the configuration layers in **`/etc/escenic/engine/`** are stored in a shared file system, so you can edit all configuration layers from one machine.)

17 Cloud Storage Configuration

You can set up the Content Engine to use cloud storage systems for storage of binary objects instead of local disks. Currently Amazon S3 is the only cloud storage provider supported To be able to use a cloud storage system you need to:

1. Set up an account with the cloud provider and get the credentials you need to access the storage. For Amazon S3, the credentials consist of a Bucket ID, an access key and a secret key.
2. Create and configure a storage provider component (**`S3FileProvider`** for Amazon S3) component in one of your configuration layers (usually the common configuration layer).
3. In the same configuration layer, create and configure one or more **`FileSystemConfiguration`** components defining how the storage provider is to be used.
4. In the same configuration layer, configure the **`Storage`** component to make use of the components you have configured.

Setting up an Amazon S3 storage account

The process of setting up an Amazon S3 account is straightforward and fully documented by Amazon, so it is not covered here. Just make sure that:

- You get an S3 Bucket with **"read-after-write" consistency** for new objects. As of February 2014 Amazon do not offer this level of data consistency in all regions, so you need to check.
- You have the following items of information about your account:
  - Bucket ID
  - Access key
  - Secret key

17.1 Create an S3FileProvider Component

1. Create a *configuration-root*/**`com/escenic/storage/aws/S3FileProvider.properties`** file, or open it if it already exists.
2. Make sure it contains the following entries:

   ```
   $class=com.escenic.storage.vfs.aws.S3FileProvider
   AWSAccessKey=your-amazon-access-key
   AWSSecretKey=your-amazon-secret-key
   ```

   where:

   - *your-amazon-access-key* is the Amazon access key for the bucket you want to use.
   - *your-amazon-secret-key* is the Amazon secret key for the bucket you want to use.

Amazon provide S3 storage in a number of different regions around the world, and the number of regions is growing. Some of the newer regions (currently the regions China (Beijing) and EU (Frankfurt) require the use of a particular signature type (Amazon Signature Version 4), while older regions also accept an older signature type. For regions that require Amazon Signature Version 4, you must include a third property in your **`S3FileProvider.properties`** file:

```
regions = region-name
```

where *region-name* is the region's official Amazon region identifier — (**CN_NORTH_1** or **EU_CENTRAL_1** in the case of China (Beijing) and EU (Frankfurt).

For all other regions no **regions** property is required at present. It can be expected, however, that more regions may be made available in the future. For up-to-date information on which regions require Amazon Signature Version 4, see http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingAWSSDK.html#specify-signature-version. For a complete list of Amazon regions and their identifiers, see http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/regions/Regions.html.

17.2 Create FileSystemConfiguration Components

You can use your cloud storage for storing all binary objects, or you can choose to store only selected types of binary objects in the cloud. You can also choose to store different types of objects in different cloud locations, in which case you will need to create more than one **FileSystemConfiguration** component. Commonly used configurations are:

- All binary objects are stored in the cloud
- Videos are stored in the cloud, all other types of binary object are stored locally. This is a common requirement since version 3.0+ of the Escenic Video plug-in requires all videos to be stored in Amazon S3.
- Video key frames are stored in a separate **read-only** file system in the cloud.

> A **read-only** file system is one where the Content Engine is not responsible for writing files to the cloud location. It simply records the file locations so that they can be retrieved when required. Read-only file systems are currently only used in combination with the Video plug-in. The Amazon Elastic Transcoder can generate images called key frames, which it stores in the cloud. A read-only file system is set up to point directly to this location so that the key frame files can be made known to the Content Engine without copying them to a new location in the cloud. For further information, see the Video plug-in documentation.

To create a **FileSystemConfiguration** component:

1. Create a *configuration-root*/**com/escenic/storage/filesystems/***component-name*.**properties** file. *component-name* should describe the purpose of the file system you are configuring. If you are going to use the file system to store videos, for example, then you might call your component **VideoFileSystemConfiguration**.
2. Make sure the file contains at least the following entries:

   ```
   $class=com.escenic.storage.FileSystemConfiguration
   name=file-system-name
   baseURI=scheme://path
   mimeTypes=mime-type-selectors
   ```

   where:

   file-system-name

   Is an identifier for the file system: **video**, for example, if you are going to use the file system to store videos.

> Once you have started to use a **`FileSystemConfiguration`**, you **must not** change its name. If you do so then the Content Engine will not be able to locate the files stored in it.

scheme
Is the URL scheme name: **`s3`**, for example. You will use this name again when you configure the **`Storage`** component (see section 17.3).

mime-type-selectors
A comma-separated list of MIME type specifications for selecting the file types to be stored in this file system. For example:

`*` to select all file types
`video/*` to select all video file types
`video/mp4` to select only MP4 videos

Any binary files that are **not** selected may be selected by a different **`FileSystemConfiguration`** (if you have defined others). See section 17.3 for more about how this works. Any files that are not selected by any **`FileSystemConfiguration`** will be stored in the default local binary file store.

3. You should also add a **`localCacheDirectory`** entry to the file:

```
localCacheDirectory=cache-path
```

where *cache-path* is the path of a local temporary folder (**`/tmp/storage/cache/video/`**, for example) to be used as a cache. Since media files tend to be large, you should choose a location with plenty of available space. Note that if you don't specify this property, then a cache is created in your Tomcat folder, which may not have sufficient available space.

4. For a file system that will be used to hold video key frames created by the Amazon Elastic Transcoder, add the following property:

```
readOnly=true
```

17.3 Configure the Storage Component

1. Open *configuration-root*/**`com/escenic/storage/Storage.properties`** for editing.
2. Add the following property to register your S3FileProvider component:

```
fileProvider.scheme=./aws/S3FileProvider
```

where:

- *scheme* is the URL scheme name (**`s3`**, for example). It must match the scheme name you have used in the **`baseURI`** property values in your **`FileSystemConfiguration`** components.

3. Add the following property to register the **`FileSystemConfiguration`**s you want to use.

```
fileSystemConfigurations=configuration-path[, configuration-path[,...]]
```

The value of this property is a comma-separated list of paths to the **`FileSystemConfiguration`** components you have configured. For example:

```
fileSystemConfigurations=./filesystems/VideoFileSystemConfiguration, ./
filesystems/BinaryFileSystemConfiguration
```

Note that the order in which you specify the components is significant. Assuming that **`VideoFileSystemConfiguration`** is configured with the MIME type selector **`video/*`** and **`BinaryFileSystemConfiguration`** is configured with the MIME type selector **`*`** then the above setting will work as intended: any video files will be caught by the **`video/*`** selector and stored in the video file system. All other files will not be selected, and will instead be caught by the **`*`** selector and stored in the generic binary file system. If the components were specified in the reverse order, however, then all files would be selected by the **`*`** selector and nothing would ever be stored in the video file system.

If none of the components specified in the list specifies a **`*`** selector, then any binary files not selected by one of the components in the list will be stored in the default local binary file store.

> The component sequence only matters when **storing** binary files, so the position of a read-only component in the list has no significance - you can put a read-only component in any position.

Your remote storage set-up is now complete.

> If you want to revert to using local storage for all files, you can do so by simply removing the **`fileProvider`**`.`*scheme* and **`fileSystemConfigurations`** properties from this file. You will also need to download any binary files that have been stored in the bucket and return them to the appropriate location in your local file system.

18 Image-related Settings

The Escenic Content Engine has a number of settings related to image handing. This chapter contains either descriptions of those settings, or links to where they are described elsewhere in the documentation.

18.1 Image Upload Size Limits

The Content Engine has two parameters that you can use to limit the size of uploaded images:

`maxImageSize`
: A maximum image size, specified in pixels. Uploaded images that exceed this limit are down-sampled to bring them under the limit. This setting is disabled by default but can be enabled at any time by defining a value.

`maxFileSize`
: An absolute file size image specified in bytes. Uploaded images that exceed this limit are discarded. The default setting is `1000000000` (1GB).

The limits apply to all images, however they are uploaded (from Content Studio, via the syndication subsystem, the API or the web service).

The limits can be configured separately for different image types by adding properties files to one of your configuration layers (section 4.1) and specifying the required property values:

- *configuration-root*`/com/escenic/storage/binaryfactory/PngProcessorFactory` for PNG images
- *configuration-root*`/com/escenic/storage/binaryfactory/JpegProcessorFactory` for JPEG images
- *configuration-root*`/com/escenic/storage/binaryfactory/OtherImageProcessorFactory` for other image types

Note that no `maxImageSize` value is set for other image types since down-sampling is not possible for unknown image types.

18.2 Image Editing Size Limit

You can prevent Content Studio from enabling editing functions if an image is over a specified size. For details, see section 14.3.1.

18.3 Image Representation Size Limit

The Content Engine has an image representation size limit (in order to avoid out-of-memory errors) which you can modify. For details, see Limiting Representation Size.

18.4 Image Quality

You can set a parameter that governs the quality of the images displayed in Content Studio. For details, see com.escenic.image.quality.