CoreMedia Content Cloud

Operations Basics

CoreMedia

CoreMedia GmbH

Altes Klöpperhaus, 5. OG

Rödingsmarkt 9

20459 Hamburg

March 14, 2024 (Release 2310)

List of Figures

List of Tables

List of Examples

1. Introduction

CoreMedia CMS is a content management system for easy and convenient creation and administration of up-to-date content, interactive features and personalized web pages.

For this purpose, CoreMedia provides an environment for online editorial workflow processes. Users can simultaneously create and edit content and so conveniently maintain a website. Integration of contents from print editorial systems, office applications and news agencies (dpa, SID, Reuters, etc.) is possible via import mechanisms. The versatile *CoreMedia Content Application Engine (CAE)* delivers content to the Internet and creates various export formats.

Configuration and operation of the different CoreMedia applications is described in the correspondent manuals. This manual describes some overall tasks and knowledge that is important for all the applications.

- An overview of the architecture of the *CoreMedia CMS* system in Chapter 2, *Component Overview* [14]
- Some basics about the operating environments used by *CoreMedia Content Cloud* are described in Chapter 3, *System Requirements* [20].
- the administration essentials of *CoreMedia CMS* for example how to start the applications, in Chapter 4, *Basics of Operation* [24]

1.1 Audience

This manual is dedicated to administrators and developers of *CoreMedia CMS* installations. They will find descriptions of all tasks necessary for installation, configuration and operation of a CoreMedia system.

1.2 Typographic Conventions

CoreMedia uses different fonts and types in order to label different elements. The following table lists typographic conventions for this documentation:

Element	Typographic format	Example
Source code Command line entries Parameter and values Class and method names Packages and modules	Courier new	`cm systeminfo start`
Menu names and entries	Bold, linked with |	Open the menu entry **Format|Normal**
Field names CoreMedia Components Applications	Italic	Enter in the field *Heading* The *CoreMedia Component* Use *Chef*
Entries	In quotation marks	Enter "On"
(Simultaneously) pressed keys	Bracketed in "<>", linked with "+"	Press the keys <Ctrl>+<A>
Emphasis	Italic	It is *not* saved
Buttons	Bold, with square brackets	Click on the **[OK]** button
Code lines in code examples which continue in the next line	\	`cm systeminfo \` `-u user`

Table 1.1. Typographic conventions

In addition, these symbols can mark single paragraphs:

Pictograph	Description
	Tip: This denotes a best practice or a recommendation.
	Warning: Please pay special attention to the text.
	Danger: The violation of these rules causes severe damage.

Table 1.2. Pictographs

1.3 CoreMedia Services

This section describes the CoreMedia services that support you in running a CoreMedia system successfully. You will find all the URLs that guide you to the right places. For most of the services you need a CoreMedia account. See Section 1.3.1, “Registration” [5] for details on how to register.

NOTE

CoreMedia User Orientation for CoreMedia Developers and Partners

Find the latest overview of all CoreMedia services and further references at:

http://documentation.coremedia.com/new-user-orientation

- Section 1.3.1, “Registration” [5] describes how to register for the usage of the services.
- Section 1.3.2, “CoreMedia Releases” [6] describes where to find the download of the software.
- Section 1.3.3, “Documentation” [7] describes the CoreMedia documentation. This includes an overview of the manuals and the URL where to find the documentation.
- Section 1.3.4, “CoreMedia Training” [10] describes CoreMedia training. This includes the training calendar,the curriculum and certification information.
- Section 1.3.5, “CoreMedia Support” [10] describes the CoreMedia support.

1.3.1 Registration

In order to use CoreMedia services you need to register. Please, start your initial registration via the CoreMedia website. Afterwards, contact the CoreMedia Support (see Section 1.3.5, “CoreMedia Support” [10]) by email to request further access depending on your customer, partner or freelancer status so that you can use the CoreMedia services.

1.3.2 CoreMedia Releases

Downloading and Upgrading the Blueprint Workspace

CoreMedia provides its software as a Maven based workspace. You can download the current workspace or older releases via the following URL:

https://releases.coremedia.com/cmcc-11

Refer to our Blueprint Github mirror repository for recommendations to upgrade the workspace either via Git or patch files.

> NOTE
>
> If you encounter a 404 error then you are probably not logged in at GitHub or do not have sufficient permissions yet. See Section 1.3.1, "Registration" [5] for details about the registration process. If the problems persist, try clearing your browser cache and cookies.

Maven artifacts

CoreMedia provides parts of its release artifacts via Maven under the following URL:

https://repository.coremedia.com

You have to add your CoreMedia credentials to your Maven settings file as described in section Section 3.1, "Prerequisites" in *Blueprint Developer Manual* .

npm packages

CoreMedia provides parts of its release artifacts as npm packages under the following URL:

https://npm.coremedia.io

Your pnpm client first needs to be logged in to be able to utilize the registry (see Section 3.1, "Prerequisites" in *Blueprint Developer Manual*).

License files

You need license files to run the CoreMedia system. Contact the support (see Section 1.3.5, "CoreMedia Support" [10]) to get your licences.

1.3.3 Documentation

CoreMedia provides extensive manuals, how-tos and Javadoc as PDF files and as online documentation at the following URL:

https://documentation.coremedia.com

The manuals have the following content and use cases:

Manual	Audience	Content
Adaptive Personalization Manual	Developers, architects, administrators	This manual describes the configuration of and development with *Adaptive Personalization*, the CoreMedia module for personalized websites. You will learn how to configure the GUI used in *CoreMedia Studio*, how to use predefined contexts and how to develop your own extensions.
Analytics Connectors Manual	Developers, architects, administrators	This manual describes how you can connect your CoreMedia website with external analytic services, such as Google Analytics.
Blueprint Developer Manual	Developers, architects, administrators	This manual gives an overview over the structure and features of *CoreMedia Content Cloud*. It describes the content type model, the *Studio* extensions, folder and user rights concept and many more details. It also describes administrative tasks for the features. It also describes the concepts and usage of the project workspace in which you develop your CoreMedia extensions. You will find a description of the Maven structure, the virtualization concept, learn how to perform a release and many more.
Connector Manuals	Developers, administrators	This manuals gives an overview over the use cases of the eCommerce integration. It describes the deployment of the Commerce Connector and how to connect it with the CoreMedia and eCommerce system.
Content Application Developer Manual	Developers, architects	This manual describes concepts and development of the *Content Application Engine (CAE)*. You will learn how to write JSP or Freemarker templates that access the other CoreMedia modules and use the sophisticated caching mechanisms of the CAE.

Manual	Audience	Content
Content Server Manual	Developers, architects, administrators	This manual describes the concepts and administration of the main CoreMedia component, the *Content Server*. You will learn about the content type model which lies at the heart of a CoreMedia system, about user and rights management, database configuration, and more.
Deployment Manual	Developers, architects, administrators	This manual describes the concepts and usage of the CoreMedia deployment artifacts. That is the deployment archive and the Docker setup. You will also find an overview of the properties required to configure the deployed system.
Elastic Social Manual	Developers, architects, administrators	This manual describes the concepts and administration of the *Elastic Social* module and how you can integrate it into your websites.
Frontend Developer Manual	Frontend Developers	This manual describes the concepts and usage of the Frontend Workspace. You will learn about the structure of this workspace, the CoreMedia themes and bricks concept, the CoreMedia Freemarker facade API, how to develop your own themes and how to upload your themes to the CoreMedia system.
Headless Server Developer Manual	Frontend Developers, administrators	This manual describes the concepts and usage of the *Headless Server*. You will learn how to deploy the Headless Server and how to use its endpoints for your sites.
Importer Manual	Developers, architects	This manual describes the structure of the internal CoreMedia XML format used for storing data, how you set up an *Importer* application and how you define the transformations that convert your content into CoreMedia content.
Multi-Site Manual	Developers, Multi-Site Administrators, Editors	This manual describes different otions to desgin your site hierarchy with several languages. It also gives guidance to avoid common pitfalls during your work with the multi-site feature.

Manual	Audience	Content
Operations Basics Manual	Developers, administrators	This manual describes some overall concepts such as the communication between the components, how to set up secure connections, how to start application.
Search Manual	Developers, architects, administrators	This manual describes the configuration and customization of the *CoreMedia Search Engine* and the two feeder applications: the *Content Feeder* and the *CAE Feeder*.
Site Manager Developer Manual	Developers, architects, administrators	This manual describes the configuration and customization of *Site Manager*, the Java based stand-alone application for administrative tasks. You will learn how to configure the *Site Manager* with property files and XML files and how to develop your own extensions using the *Site Manager API*. The Site Manager is deprecated for editorial work.
Studio Developer Manual	Developers, architects	This manual describes the concepts and extension of *CoreMedia Studio*. You will learn about the underlying concepts, how to use the development environment and how to customize *Studio* to your needs.
Studio User Manual	Editors	This manual describes the usage of *CoreMedia Studio* for editorial and administrative work. It also describes the usage of the *Adaptive Personalization* and *Elastic Social* GUI that are integrated into *Studio*.
Studio Benutzerhandbuch	Editors	The Studio User Manual but in German.
Supported Environments	Developers, architects, administrators	This document lists the third-party environments with which you can use the CoreMedia system, Java versions or operation systems for example.
Unified API Developer Manual	Developers, architects	This manual describes the concepts and usage of the *CoreMedia Unified API*, which is the recommended API for most applications. This includes access to the content repository, the workflow repository and the user repository.

Manual	Audience	Content
Utilized Open Source Software & 3rd Party Licenses	Developers, architects, administrators	This manual lists the third-party software used by CoreMedia and lists, when required, the licence texts.
Workflow Manual	Developers, architects, administrators	This manual describes the *Workflow Server*. This includes the administration of the server, the development of workflows using the XML language and the development of extensions.

Table 1.3. CoreMedia manuals

If you have comments or questions about CoreMedia's manuals, contact the Documentation department:

Email: documentation@coremedia.com

1.3.4 CoreMedia Training

CoreMedia's training department provides you with the training for your CoreMedia projects either live online, in the CoreMedia training center or at your own location.

You will find information about the CoreMedia training program, the training schedule and the CoreMedia certification program at the following URL:

http://www.coremedia.com/training

Contact the training department at the following email address:

Email: training@coremedia.com

1.3.5 CoreMedia Support

CoreMedia's support is located in Hamburg and accepts your support requests between 9 am and 6 pm MET. If you have subscribed to 24/7 support, you can always reach the support using the phone number provided to you.

To submit a support ticket, track your submitted tickets or receive access to our forums visit the CoreMedia Online Support at:

http://support.coremedia.com/

Do not forget to request further access via email after your initial registration as described in Section 1.3.1, "Registration" [5]. The support email address is:

Email: support@coremedia.com

Create a support request

Support request

CoreMedia systems are distributed systems that have a rather complex structure. This includes, for example, databases, hardware, operating systems, drivers, virtual machines, class libraries and customized code in many different combinations. That's why CoreMedia needs detailed information about the environment for a support case. In order to track down your problem, provide the following information:

- Which CoreMedia component(s) did the problem occur with (include the release number)?
- Which database is in use (version, drivers)?
- Which operating system(s) is/are in use?
- Which Java environment is in use?
- Which customizations have been implemented?
- A full description of the problem (as detailed as possible)
- Can the error be reproduced? If yes, give a description please.
- How are the security settings (firewall)?

In addition, log files are the most valuable source of information.

Support checklist

To put it in a nutshell, CoreMedia needs:

1. a person in charge (ideally, the CoreMedia system administrator)
2. extensive and sufficient system specifications
3. detailed error description
4. log files for the affected component(s)
5. if required, system files

Log files

An essential feature for the CoreMedia system administration is the output log of Java processes and CoreMedia components. They're often the only source of information for error tracking and solving. All protocolling services should run at the highest log level that is possible in the system context. For a fast breakdown, you should be logging at debug level. See Section 4.7, "Logging" [49] for details.

Which Log File?

In most cases at least two CoreMedia components are involved in errors: the *Content Server* log files together with the log file from the client. If you know exactly what the problem is, solving the problem becomes much easier.

Where do I Find the Log Files?

By default, application containers only write logs to the console output but can be accessed from the container runtime using the corresponding command-line client.

For the *docker* command-line client, logs can be accessed using the **docker logs** command. For a detailed instruction of how to use the command, see docker logs. Make sure to enable the timestamps using the `--timestamps` flag.

```
docker logs --timestamps <container>
```

For the *kubectl* command-line client in a Kubernetes environment you can use the **kubectl logs** command to access the logs. For a detailed instruction of how to use the command, see kubectl logs. Make sure to enable the timestamps using the `--timestamps` flag.

```
kubectl logs --timestamps <pod>
```

1.4 Changelog

In this chapter you will find a table with all major changes made in this manual.

Section	Version	Description

Table 1.4. Changes

2. Component Overview

Overview and deployment

CoreMedia CMS is a distributed web content management system (WCMS) for creation, management and delivery of context dependent content. Most of the applications of CoreMedia CMS are deployed as web applications in a servlet container. Only the *Site Manager* and the server utilities are deployed as stand-alone applications. All applications can be deployed into the Cloud.

With *CoreMedia Content Cloud* you do not get a program to install and run, but a workspace to develop within, to build and to deploy artifacts from.

Security

The communication between all applications can be secured. See Section 4.4, "Communication between the System Applications" [30] for details.

Logging and Monitoring

All applications of *CoreMedia CMS* use Logback for logging. See Section 4.7, "Logging" [49] for details. By default, all CoreMedia applications register relevant resources via JMX as MBeans for management and monitoring purposes. So, you can use a common JMX client such as JConsole to change or check the configuration, to start tasks or to get statistic data. If you only want to have a look at the configured JMX parameter and its values, you can simply use the CoreMedia utility *jmxdump*, which simply prints out this information, as described in Section 3.13.2.10, "JMXDump" in *Content Server Manual*.

2.1 Architectural Overview

Figure 2.1, “Architectural Overview” [15] shows a deployment of *CoreMedia CMS*.

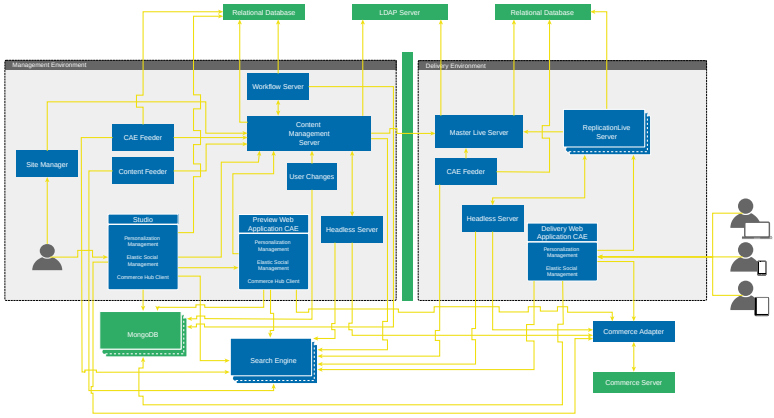

Figure 2.1. Architectural Overview

If you want to have an overview of the default ports used in CoreMedia, check out Section 4.4.1, “Default Application Ports” [31].

2.2 Communication of Components

A CoreMedia system is separated into a management environment where the editors are working and a delivery environment where the customers access the website. The environments can be separated by a firewall for security reasons. The applications communicate via HTTP and CORBA, see Section 4.4, "Communication between the System Applications" [30] for details. Table 2.1, "CoreMedia applications" [16] shows all applications of *CoreMedia CMS*, describes what they do, if there are multiple instances and with which applications they communicate:

Application	Purpose	Multiple Instances	Communicates with
Content Management Server (For more information, check out: Section 2.1, "The Content Server" in *Content Server Manual*9	Manages the content in the Management Environment and publishes content to the *Master Live Server*.	No	• All clients • Publishes content to the Master Live Server • External relational database • Search Engine
Master Live Server (For more information, check out: Section 2.1, "The Content Server" in *Content Server Manual*)	Manages the content in the Delivery Environment	Multiple instances when Multi-Master is used	• All clients. • External relational database
Replication Live Server (For more information, check out: Section 2.2, "Replication Live Servers" in *Content Server Manual*)	Serves content to the CAEs	Multiple instances can be attached to one *Master Live Server*	• Content Application Engine • External relational database
Workflow Server (For more information, check out: Chapter 2, *Overview of CoreMedia Workflow* in *Workflow Manual*)	Executes workflows	No	Content Management Server
Site Manager (For more information, check out: Site Manager Developer Manual)	Management tool for workflows and users.	Yes	• Content Management Server • Workflow Server

Application	Purpose	Multiple Instances	Communicates with
Studio (For more information, check out: Section 2.1, "Architecture" in *Studio Developer Manual*)	Content editing and management. Hosts management extensions for *Elastic Social* and *Adaptive Personalization*.	One web application	• Content Management Server • Search Engine • Workflow Server • MongoDB • External relational database
Search Engine (For more information, check out: Chapter 3, *Search Engine* in *Search Manual*)	Indexes content and provides searches functionality.	Yes.	• Content Management Server • Content Feeder • CAE Feeder • Studio • Content Application Engine
CAE Feeder (For more information, check out: Chapter 5, *Searching for CAE Content Beans* in *Search Manual*)	Feeds content beans into the *Search Engine*	Multiple instances possible, for example when reindexing.	• Content Management Server • Search Engine • External relational database
Content Feeder (For more information, check out: Chapter 4, *Searching for Content* in *Search Manual*)	Serves content to the *Search Engine*	Multiple instances possible, for example when reindexing.	• Content Management Server • Search Engine
Content Application Engine (For more information, check out: Chapter 2, *Overview* in *Content Application Developer Manual*)	Serves sites to the customer. Hosts *Elastic Social* and *Adaptive Personalization* extension.	Multiple instances can be attached to one *Master Live Server* or *Replication Live Server*	• Content Server • MongoDB database for *Elastic Social* • Search Engine • Custom external systems

Application	Purpose	Multiple Instances	Communicates with
Importer (For more information, check out: Chapter 3, *Administration And Operation* in *Importer Manual*)	Imports content into the *Content Management Server*.	Yes	• Content Management Server

Table 2.1. CoreMedia applications

2.3 Third-Party Requirements

Third-party software

As shown in Figure 2.1, "Architectural Overview" [15] *CoreMedia CMS* requires some third-party software for operation, which is not delivered with *CoreMedia CMS*. In general, the following software has to be installed:

- A Java installation.
- A relational database for the content storage.
- A MongoDB database for *Elastic Social*.
- A browser for *CoreMedia Studio*.

Most of the applications require a servlet container as a runtime environment.

In addition, you can run *CoreMedia CMS* with an LDAP server. Find a list of all supported environments at https://documentation.coremedia.com/cmcc-11.

Installation

3. System Requirements

Use only recommended systems

A CoreMedia system has to rely on several (third-party) software components, for proper operation. CoreMedia tests *CoreMedia CMS* with the most common combinations used by our customers and distinguishes between two levels of approved infrastructure components:

- *Certified level*
  Certified infrastructure components are extensively tested to work with the *CoreMedia CMS* system. Every infrastructure component approved with the first final CMS Release is certified. It is recommended to use these components for productive systems.
- *Supported level*
  Supported infrastructure components will also work with CoreMedia applications but they are tested less exhaustively, because they are released after the first final CMS Release. They also can be used for productive systems. Refer to the `notes.html` file for announcements of additionally supported environments or the reference of this manual.

Note: the state "deprecated" is also used on occasion. Deprecated infrastructure components are either of certified or supported level in the current version of the *CoreMedia CMS* but do not carry official approval by CoreMedia beyond this version.

> **NOTE**
>
> All necessary security updates for approved versions, recommended by vendors of infrastructure components (such as OS, Java, database...), are supported by CoreMedia automatically. This does not apply to feature updates!

Supported operation environments

You will find the approved components in the Supported Operation Environments document on https://documentation.coremedia.com/cmcc-11. In the following sections you will find some general hints for the usage of these components:

- Java platforms in Section 3.1, “Java” [22],
- Databases in Section 3.2, “Databases” [23],

CAUTION

Please keep in mind, that the databases and application servers have only been tested in CoreMedia compliant operating environments and therefore are only approved on these platforms.

3.1 Java

The functionality of CoreMedia applications can only be guaranteed with approved platforms and corresponding Java versions. To operate *CoreMedia CMS*, run the Java platform with Java Runtime Environment (JRE) or Java Development Kit (JDK).

CAUTION

Do not run a *CoreMedia CMS* System with different Java versions. All applications have to use the same Java version.

The appropriate JREs/JDKs for the different supported platforms can be obtained from the following locations:

- for Solaris, Linux and Windows JRE/JDK can be downloaded at Oracle (http://www.oracle.com or http://www.oracle.com/technetwork/java/javase/downloads/index.html).
- the IBM JRE/JDK can be downloaded at IBM (https://www.ibm.com/developerworks/java/jdk/).

NOTE

Only use the JRE/JDK binaries listed in the Supported Environments document or further approved versions mentioned in the change notes on the documentation site. Don't use any other than the specified patch level of an JRE/JDK version! A different patch level is not supported and probably causes errors in service.

3.2 Databases

CoreMedia CMS uses repositories for data storage, therefore it requires one or more external relational databases. A correctly installed and activated database is prerequisite for the operation of *CoreMedia CMS*. How to connect the *CoreMedia* system to databases is described in detail for the different databases in the *Content Server Manual*.

NOTE

It is strongly recommended to use a UTF-8 enabled database for your *CoreMedia CMS* repository.

The databases have only been tested in CoreMedia compliant operating environments and therefore are only approved on these platforms. For all supported environments see the [Supported Environments] document at http://bit.ly/cmcc-11-supported-environments.

4. Basics of Operation

This chapter covers the fundamental principles of CoreMedia system administration and operation - an overview of the directory structure of the CoreMedia system, configuration settings for internal system communication and general aspects of operating CoreMedia applications.

4.1 Starting CoreMedia Command-Line Tools

Depending on their function the command-line tools are split into several directories, for example the tools that work with the *Content Management Server* are combined in a directory called `cms-tools`. The command-line tools are started by calling the `cm` command. On entering the command `bin/cm` without further details, an overview of the commands available in the respective directory is given.

Under Windows the command-line tools can be started with a JVM 64-bit using the `cm64.exe` application launcher which is also located in the `bin` directory.

Note that the `cm` command always changes the working directory to `COREM_HOME`, which is the base directory of the tools. Thus, if a relative path is given as a parameter (with the *`-script`* parameter in `cm sql`, for instance) it must be relative to `COREM_HOME`.

The `cm` command can use the `-nolog` option. This option overwrites the *`OUTPUT_REDIRECT`* parameter setting of the `cm.jpif` file with the empty value. Thus, all log output is written to standard out.

You will find a description of all server utilities in Section 3.13, "Server Utility Programs" in *Content Server Manual* and Section 3.5, "Workflow Server Utilities" in *Workflow Manual*. Other tools which can be started with `cm` can be found all over the manual.

Note: `<cm>-xmlimport` is a freely configurable XML importer, in which the prefix `<cm>` can be exchanged for any other desired prefix (see the Importer Manual).

```
$ cm
Usage: bin/cm application parameter*
where application is one of: approve bulkpublish cancelpublication
changepassword checklicense cleanrecyclebin cleanversions dbindex
destroy destroyversions dump dumpusers encryptpasswordproperty
encryptpasswords events groovysh ior jconsole jmxdump
killsession license migrateplacements module multisiteconverter
post-config pre-config processorusage publications
publishall publish queryapprove query querypublish recordstate
repositorystatistics republish restorestate restoreusers rules
runlevel schemaaccess search serverexport serverimport sessions
sql systeminfo tracesession unlockcontentserver usedlicenses
validate-multisite version
```

Example 4.1. Output of `cm` in the `cms-tools` directory

4.1.1 Configuration of the Start Routine with JPIF Files

Each command-line tool has its own start file with the ending ".jpif", which is executed on startup. The name of this file corresponds to the name used for starting the application with the `cm/` command (for example `cm runlevel` uses `runlevel.jpif`). You'll find these files in the `<COREM_HOME>/bin` directory.

The JPIF files for applications determine which Java class should be executed on starting the application. Further settings for the operation of the application can also be stored in this file. This file can be used to modify the Java Virtual Machine (JVM) where the application runs, while parameters can be passed to the JVM.

The following CoreMedia relevant modifications can be configured for the Java Virtual Machine in the `JAVA_VM_ARGS` section of the JPIF file:

The memory usage within the Java Virtual Machine can be configured using the explicit parameters `-Xms<size>` and `-Xmx<size>` or the relative parameters `-XX:MinRAMPercentage=<size>` and `-XX:MaxRAMPercentage=<size>`. `-Xms` and `-XX:MinRAMPercentage` specify the initial object memory size. `-Xmx` and `-XX:MaxRAMPercentage` the maximum object memory size. The memory requirement for the applications is not preconfigured and it should be sized according to the standard hardware recommendations. To size an application, you should use the default JVM command-line augmentation facade, the `JAVA_TOOL_OPTIONS` environment variable, either set globally or per process.

The ORB can be configured to use a fixed CORBA port using the parameter `com.coremedia.corba.server.port` as described in Section 4.4, "Communication between the System Applications" [30].

Furthermore, the target of the log outputs of the Java process (see Section 4.7, "Logging" [49]) can be configured with the parameter `OUTPUT_REDIRECT`.

Three JPIF files cannot be invoked directly with the `cm` command. They are executed internally:

- `pre-config.jpif` for installation depending settings. In this file, the parameter `VERBOSE` can be set to `false` to reduce JVM outputs. On a Unix system, the JVM to use is set in this file.
- `module.jpif` for general environment settings for the Java programs in the CoreMedia system.
- `post-config.jpif` for special CoreMedia JVM settings.

In general, these files need not be changed.

4.1.2 Which JVM will be used?

For command-line tools the information about the JVM to use is read from the property `JAVA_HOME` in the `pre-config.jpif` file or from the environment variable.

If `JAVA_HOME` is not set, a JVM installed in the `COREM_HOME` directory will be used as the active JVM. The installation directory of the JVM has to be located directly below these directories. For example, `<COREM_HOME>/jre`.

4.2 CoreMedia CMS Directory Structure

CoreMedia applications come either as or as Spring Boot applications or as applications using the CoreMedia proprietary application structure. The latter will be described here.

CoreMedia applications

A CoreMedia application, the *Server Utilities* for example, has the following directories:

- `./bin`: Start scripts (see Section 4.1, "Starting CoreMedia Command-Line Tools" [25]) for Unix (cm) and Windows (*cm.exe*, *cmw.exe*) as well as the start scripts of the individual CoreMedia utility programs and the *Site Manager*.
- `./lib`: Runtime resources like Java JAR files and DLLs.
- `./classes`: Optional local classes. Note: The directory does not exist in the standard installation. It can contain customer-specific extensions.
- `./config/<component>`: XML configuration files of the application.
- `./properties/corem`: *CoreMedia CMS* configuration files in Java properties format.
- `./var/log`: log files of the CoreMedia applications (see Section 4.7, "Logging" [49]).
- `./var/run`: runtime data (such as Process ID).
- `./var/tmp`: temporary data.

4.3 Configuration of CoreMedia Applications

CoreMedia server application, like the *Content Application Engine* or the *Content Management Server* for example, are deployed as a Spring Boot application JAR file and therefore follow the Spring Boot defaults for externalize configuration.

All other applications that follow the proprietary application structure, like the command-line utilities or the Site Manager, can be configured using the following instructions:

CoreMedia applications are configured with Java properties files with the ending `.properties`. The encoding is `ISO-8859-1`. Each line stores a single property with the format `key=value`. The hash sign `#` is used for labeling comments, and the backslash `\` is used as escape character.

Each application of the *CoreMedia* system has one or more relevant property files where the operation of the application can be configured.

The locations of properties files for *CoreMedia* applications are (depending on the particular application):

- `properties/corem`
- `config`

Windows Paths in Java Properties Files

When you configure a Windows paths in a property file, you have to escape a backslash with a second backslash in the path. This applies especially to paths for an importer inbox path. For more details about writing property values, see the Javadoc for the load() method in the `java.util.Properties` Java class.

4.4 Communication between the System Applications

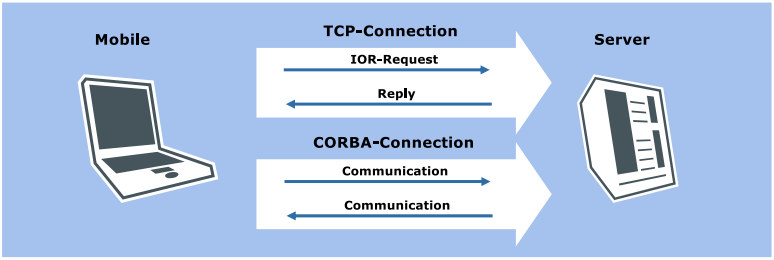

Figure 4.1. IOR inquiry and answer between CoreMedia Client and Server

CORBA is used for the communication between CoreMedia system applications. All CoreMedia applications require the IOR of the *Content Server* which they want to communicate with. The IOR of the *Content Server* will be delivered by the server via the HTTP protocol.

- All applications require the IOR of the *Content Server* with which they want to communicate.
  The URL where to get the IOR of the *Content Server* is configured with the parameter `cap.client.server.ior.url=<IOR-URL>` in the file `capclient.properties`.
  The value of the parameter is `http://<server>:<port>/ior`. Instead of <server> you have to insert the name of the computer where the server is running. Instead of <port> you have to insert the HTTP port on which the client connects to the server.
  Example: The *Content Server* host has the name productionserver and the property `cap.server.http-port` is set to `44445`. In this case, you can obtain the IOR with the following URL:
  `cap.client.server.ior.url=http://productionserver:44445/ior`

The *Content Management Server*/*Live Server* embed their own host names into the IOR which must be resolved by the client machines. If this is not possible by the client, you can configure the server to embed a numeric IP address into the IOR. To do so, set the property `com.coremedia.corba.server.host`. In the following example, the ORB is configured to embed its numeric address, by setting a system property:

```
-Dcom.coremedia.corba.server.host="10.1.3.253"
```

The Unified API takes care of detecting and cleaning up stale TCP connections at the CORBA level. This aids in reconnecting to the servers after a communication failure or a server downtime. If reconnects happen spuriously without an obvious cause, this feature can be disabled by setting the system property `com.coremedia.corba.orb.reconnect=false`.

As said before, classic CoreMedia client applications read its `capclient.properties` file to access the property `cap.client.server.ior.url` for the IOR URL of the server. Newer CAE/Spring/Unified API based clients read its Spring configuration file (`repository.xml`, `CapConnectionFactory`...) to access the server IOR. When Content Servers act as clients to access other Content Servers, they read the IOR URL from other configuration files:

- The *Content Management Server* must know the IOR of the *Master Live Server* during publication.
  The IOR URL is stored in the property `publisher.target[0].ior-url`.
- The *Replication Live Server* (when installed) has to communicate with its *Master Live Server*.
  The IOR URL is stored in the property `replicator.publicationIorUrl`.

4.4.1 Default Application Ports

Depending on the deployment setup, the ports on each application are either standardized to identical ports or to unique ports:

- Plain Spring Boot JAR files define standardized identical ports. If you plan to install services using the plain JAR files, make sure to set unique ports if multiple applications should be installed on the same host.
- The Docker images use standardized identical ports. The container abstraction ensures that there cannot be a port conflict between two containers unless both forward a port to the same port on the host.
- The deployment archive configures a unique port schema for all applications. In ????, you can find the port conventions.

4.4.2 Communication Through a Firewall

In order to communicate with the *CoreMedia Server* or *Workflow Server*, two open ports are required:

- The HTTP port to fetch the IOR
- The CORBA port for communication

In the default configuration, the CORBA port changes with every restart of the application server which is inconvenient in case of an intermediate firewall. In this case, the port can be set to a fixed value through the property `com.coremedia.corba.server.port`. In the following example, the ORB is configured to listen on port 55555, by setting a system property:

- `-Dcom.coremedia.corba.server.port=55555`

If you want to access the *Server* from "outside" a firewall and the server IP address is not directly accessible (due to network address translation for example), it is possible to establish an SSH tunnel. The tunnel forwards all traffic from the client to the server. Of course, the endpoint of the tunnel must be able to reach the server. Figure 4.2, "Schema of the SSH tunnel" [32] shows the scenario:

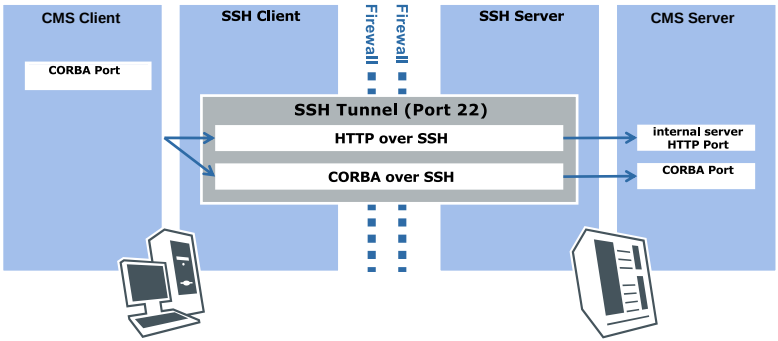

Figure 4.2. Schema of the SSH tunnel

Four parties are involved in the tunneling:

- A client <CMSClient> which cannot access the server directly.
- The client-side SSH client <SSHClient> which cannot access the *Content Server*.
- The server-side SSH server <SSHServer> which can access the *Content Server*.
- The CoreMedia Server <CMSServer>.

<CMSClient>/<SSHClient> and <CMSServer>/<SSHServer> can reside on the same machine respectively.

Two ports must be configured:

- <HTTPPort> is the HTTP port for the IOR.
- <CORBAPort> is the port for CORBA communication.

For this scenario you must do the following:

- Establish the tunnel

- Redirect client requests to the tunnel endpoint SSHClient instead of CMSServer

Proceed as follows:

1. Configure the HTTP port of the server as usual, for example, in application.properties.
2. Configure the HTTP address where to fetch the IOR of the server in the `capclient.properties` file as follows:

   `cap.client.server.ior.url=http://<SSHClient>:<HTTPPort>/ior`

3. Start a SSH server on <SSHServer>. No particular configuration is necessary.
4. Start the SSH client on <SSHClient>.
5. On a UNIX system, open the tunnel on the SSHClient with

   ```
   ssh -g -L<CORBAPort>:<CMSServer>:<CORBAPort> \
          -L<HTTPPort>:<CMSServer>:<HTTPPort> \
          <SSHServer>
   ```

   . Replace the values in angle brackets with the appropriate settings.
6. For the Windows SSH client SSH Secure Shell choose `Edit | Settings | Profile Settings | Tunneling | Incoming`. You need to make two entries. Insert as follows:
   - Type: TCP
   - Listen Port: <HTTPPort>
   - Destination Host: <CMSServer>
   - Destination Port: <HTTPPort>

   and
   - Type: TCP
   - Listen Port: <CORBAPort>
   - Destination Host: <CMSServer>
   - Destination Port: <CORBAPort>

   This will instruct ssh to forward all requests on <SSHClient>:<Port> via <SSHServer> to <CMSServer>:<Port>.

6. In order to instruct a client to contact <SSHClient> instead of <CMSServer>, you need to configure its client-side ORB with system properties.

   For command line tools, you can set system properties in the JPIF file.

You need to set the following properties, replacing <CMSServer> and <SSHClient> with the names of the appropriate computers and <CorbaPort> with the port number of the ends of the SSH tunnel:

Property Type	Property Name	Property Value
System	`com.coremedia.corba.client.redirect.original-host`	<CMSServer>
System	`com.coremedia.corba.client.redirect.redirect-host`	<SSHClient>
System	`com.coremedia.corba.client.redirect.original-port`	<CorbaPort>
System	`com.coremedia.corba.client.redirect.redirect-port`	<CorbaPort>

Table 4.1. Properties for SSH configuration

An alternative to setting up a SSH tunnel might be the use of a VPN, or SSL.

4.4.3 Binding Only a Single Network Interface

By default, both HTTP port and the CORBA port are bound to all network interfaces. For example your server might be accessible through two network cards using the IP addresses 10.1.3.253 and 10.1.3.254. For security reasons, you might want to grant access to the servers only through one of the interfaces.

Binding the HTTP port to only one single interface can be achieved by setting the property `server.address` to the corresponding IP.

For limiting the access through CORBA, too, set the following properties when starting the *Content Management Server* and the *Workflow Server*:

Property Type	Property Name	Property Value
System	`com.coremedia.corba.server.host`	<IpAddress>

Property Type	Property Name	Property Value
System	`com.coremedia.corba.server.single-ip`	<IpAddress>

Table 4.2. Properties for Single IP configuration

Replace <IpAddress> by the IP address of the network interface to bind, for example 10.1.3.253. If you want to secure this connection via SSL, proceed with the next section.

4.4.4 Encrypting CORBA Communication Using SSL

In a standard CoreMedia installation, session handles and content are transmitted in clear text across the network between client and server. This is usually not a problem when the editorial workplaces and the servers reside in the same trusted network. However, for secure remote access, encrypted communication is sometimes required.

If SSH tunneling is not an option, alternatively a Secure Socket Layer (SSL) connection can be used for the CORBA communication between CoreMedia applications.

The setup is slightly more complex than in the SSH case, because the certificate handling has to be administered explicitly for Java's SSL implementation, and because the port mapping has to be specified in CoreMedia configuration files.

In the following example it is assumed that communication has to be encrypted between a *Site Manager* on one side, and the *Content Server* and *Workflow Server* on the other side.

NOTE

In this example, the port numbers from table Table 4.3, “Example SSL Ports” [35] are used. You may want to use different port numbers for your deployment.

Server	Clear-Text Port	SSL Port
Content Server	14300	14443
Workflow Server	14305	14445

Table 4.3. Example SSL Ports

The servers open an SSL Port in addition to the clear-text port. This allows the same server to be accessed using clear text communication from within a trusted network, and using SSL from outside. When a client is configured to use SSL, not a single byte will be sent to the clear text port, which may be blocked from outside access by a firewall.

Note that the server's HTTP port will have to be accessible to clients, for example to retrieve the IOR.

Enable SSL Encryption

Enabling SSL encryption for CORBA communication requires the following steps:

1. Create key stores for *Content Server*, *Workflow Server* and clients.
2. Prepare the *Content Server* for SSL communication
3. Prepare the *Workflow Server* for SSL communication
4. Prepare the client for SSL communication.
5. Restart all three applications
6. Verify SSL communication

Create key stores

Create **key stores** which will later be distributed to the servers and clients. Consult your JDK documentation for further details about the **keytool** command.

1. Create self-signed server keys for Content Server and Workflow Server

```
keytool -genkey -alias contentserver -v -keyalg RSA \
   -keystore contentserver.keystore
keytool -genkey -alias workflowserver -v -keyalg RSA \
   -keystore workflowserver.keystore
```

2. Export the server's public keys from their key stores:

```
keytool -export -rfc -keystore contentserver.keystore \
   -alias contentserver -file contentserver.public-key
keytool -export -rfc  -keystore workflowserver.keystore \
   -alias workflowserver -file workflowserver.public-key
```

Prepare the *Content Server* for SSL communication

1. Add the following properties to the content server's configuration:

Property Name	Property Value
`com.coremedia.corba.server.port`	14300

Property Name	Property Value
`com.coremedia.corba.server.ssl.ssl-port`	14443
`com.coremedia.corba.server.ssl.keystore`	<path to contentserver.keystore>
`com.coremedia.corba.server.ssl.passphrase`	<mypassword>

Table 4.4. Properties for Content Server SSL configuration

2. Place the `contentserver.keystore` in the location defined by the *`com.coremedia.corba.server.ssl.keystore`* property.

Prepare the *Workflow Server* for SSL communication

1. Add the following properties to the workflow server's configuration:

Property Name	Property Value
`com.coremedia.corba.server.port`	14305
`com.coremedia.corba.server.ssl.ssl-port`	14445
`com.coremedia.corba.server.ssl.keystore`	<path to workflowserver.keystore>
`com.coremedia.corba.server.ssl.passphrase`	<mypassword>

Table 4.5. Properties for Workflow Server SSL configuration

2. Place the `workflowserver.keystore` in the location defined by the *`com.coremedia.corba.server.ssl.keystore`* property of the workflow server.

The following two steps are optional and are limited to rare cases, in which SSL encrypted communication may also be required between workflow server and content server.

3. In this case, you should add the content server's key to the workflow server's **key store**, and configure the workflow server as an SSL client. Run the following command:

```
keytool -import -alias contentserver -keystore \
  workflowserver.keystore -file contentserver.public-key
```

4. In addition to the above, set the following client system properties during invocation of the *Workflow Server*:

Property Name	Property Value
`com.coremedia.corba.client.ssl.clear-text-ports`	14300
`com.coremedia.corba.client.ssl.ssl-ports`	14443
`com.coremedia.corba.client.ssl.keystore`	<path to workflowserver.keystore>
`com.coremedia.corba.client.ssl.passphrase`	<mypassword>

Table 4.6. Properties for Workflow to Content Server SSL configuration

Preparing a client ORB for SSL communication

All CoreMedia clients use CORBA to communicate with the servers.

1. Import the servers' public keys to the clients's key store:

```
keytool -import -alias contentserver \
  -keystore editor.keystore -file contentserver.public-key
keytool -import -alias workflowserver \
  -keystore editor.keystore \
  -file workflowserver.public-key
```

2. Configure the ORB for SSL by setting the properties from Table 4.7, “Properties for Client ORB SSL configuration” [38].

Property Name	Property Value
`com.coremedia.corba.client.ssl.clear-text-ports`	14300,14305
`com.coremedia.corba.client.ssl.ssl-ports`	14443,14445

Property Name	Property Value
`com.coremedia.corba.client.ssl.keystore`	<path to editor.keystore>
`com.coremedia.corba.client.ssl.passphrase`	<mypassword>

Table 4.7. Properties for Client ORB SSL configuration

The comma separated values of the `clear-text-ports` and the `ssl-ports` properties must match. They must have the same length, and the n-th value of each property refers to the same component. In this example the first values, 14300 and 14443, denote the content server, and the second values, 14305 and 14445 belong to the workflow server.

Prepare the Site Manager for SSL Communication

1. Create a keystore for the Site Manager.
2. Import the servers' public keys to the Site Manager's *key store*:

```
keytool -import -alias contentserver \
  -keystore editor.keystore -file contentserver.public-key
keytool -import -alias workflowserver \
  -keystore editor.keystore \
  -file workflowserver.public-key
```

2. Add the following lines to `bin/editor.jpif`.

```
JAVA_VM_ARGS="$JAVA_VM_ARGS
-Dcom.coremedia.corba.client.ssl.clear-text-ports=14300,14305"
JAVA_VM_ARGS="$JAVA_VM_ARGS
-Dcom.coremedia.corba.client.ssl.ssl-ports=14443,14445"
JAVA_VM_ARGS="$JAVA_VM_ARGS
-Dcom.coremedia.corba.client.ssl.keystore=properties/corem/editor.keystore"
JAVA_VM_ARGS="$JAVA_VM_ARGS
-Dcom.coremedia.corba.client.ssl.passphrase=mypassword"
```

3. Place the `editor.keystore` in `properties/corem/` of your installation.

CAUTION

Though stated in the examples, it is not recommended to place the `editor.keystore` at any publicly accessible place. This is only intended for testing and development. For productive use, an official key should be deployed with every Unified API installation on the client machines. Another possible way would be to download the key store via HTTPS using a certificate that is already present on the workplace computers.

Restart *Workflow Server*, *Content Server*, and clients.

Restart all servers by restarting the servlet container where they are deployed.

Verify SSL communication

Verify SSL communication by searching the applications' logs for error messages, and by using *netstat* or *lsof*. Under Solaris, using the port numbers in this example, you could use the command:

```
netstat -e -a -p|grep ":14[34]"
```

It should show that before starting the Site Manager, the server is listening on port 14443/14445 (which are the SSL ports) and 14300/14305 (the clear text ports). After the Site Manager is started and a user has logged in, a connection should be established on port 14443/14445 (and *not* 14300/14305) towards the client's machine. Note that other applications might continue to connect to the clear text ports.

4.4.5 Preparing Spring Boot applications for HTTPS Connection

HTTPS is a variant of HTTP which enables encrypted data transmission between server and client. It is therefore recommended, that you create the servlet container client (*CAE*) connection via HTTPS. This chapter describes how you create a key and how you configure Tomcat to use this key.

4.4.5.1 Creating a Key

In order to connect client and server application via HTTPS you have to generate a key for the servlet container. This key is sent from server to client with each query of the client to the server. The client decides whether the sender of the key is trustworthy with every single request.

Creation of the key

The tool for creating the key is supplied with the JDK. You create the key with the following entries:

1. Enter the following command:

```
<java-home>/bin/keytool -genkey -alias spring-boot \
  -keyalg RSA \
  -storetype PKCS12 \
  -keysize 2048 \
  -keystore /example/coremedia/coremedia.keystore \
```

   In this way you call the program `keytool` in the directory `<java-home>/bin`. You initiate creation of the key (`-genkey`) with the alias name (`-alias spring-boot`). A key is created according to the RSA algorithm. The key is saved in the `-keystore` file `/example/coremedia/coremedia.keystore` (here you can enter your own path/name). If you already have a key store file, you must enter the location of this file.

2. At the next input request, enter a password. If you want to save the key in an already existing key store, you must enter the password of this file.
3. At the next input request, enter the name of the server (the entry given below is an example).

   What are your first and last name?

   ```
   [Unknown]: webserver.coremedia.com
   ```

4. Confirm the following input requests with <Return>, until you are asked to confirm the correctness of the previous entries.
5. Enter "y" and <Return> to confirm the previous entries. You can cancel by entering <Return>.

   After a short pause, you are asked for the "key password for < Spring Boot>".

6. Enter the password you have defined in step 2 for your newly created key with the alias "tomcat".

Now, you have finished key creation.

4.4.5.2 Configuring Spring Boot

With Spring Boot configuring SSL can be done completely by setting a set of properties. For a complete reference of properties available, see common Spring application prop-

erties and look for `server.ssl.` prefix. For the current example, configure the properties below:

```
server.ssl.enabled=true
server.ssl.key-alias=spring-boot
server.ssl.key-password=changeit
server.ssl.key-store=/example/coremedia/coremedia.keystore
server.ssl.key-store-type=PKCS12
server.ssl.key-store-password=changeit
```

4.4.6 Troubleshooting

Applications do not respond on request. The CPU load of the applications is high, the thread dump shows threads which use `nio` classes.

Possible cause:

Problems with the CORBA ORB.

Possible Solution:

Add the following ORB property as a system property for the affected applications:

```
-Dcom.sun.corba.ee.transport.ORBUseNIOSelectToWait=false
```

4.5 Collaborative Components

CoreMedia offers tools for collaboration between editors in Studio. Collaboration means sharing content, collaborating by publishing and translating content, assigning tasks to users, and notifying editors about recent actions with their content.

4.5.1 Overview

The following components provide collaborative features in CoreMedia Studio:

- Studio Control Room Plugin
- Notifications Studio Plugin
- User Changes Web Application
- Extensions of the *Workflow Server*

4.5.2 Deployment

The default deployment of CoreMedia's collaborative components is with a MongoDB database. When deployed with a MongoDB database, configure the collaborative components to connect to your MongoDB instance using the configuration properties given below.

Property	Example	Description
`mongodb.client-uri`	`mongodb://<Username>:<Password>@<Host>:<Port>/`	The URL of the MongoDB to connect to. Replace `<Username>`, `<Password>`, `<Host>` and `<Port>` with the appropriate values of the MongoDB installation. Add this property to the `WEB-INF/application.properties` file of *Studio*, *User Changes* and *Workflow Server* applications, and let it point to your *MongoDB*.
`mongodb.prefix`	`<prefix>`	When the collaborative components persist collaboration data to a *MongoDB* database, the default name of its database is prefixed by `blueprint`. To configure a different database name prefix, add this property to `WEB-INF/ap`

Property	Example	Description
		`plication.properties` files of *Studio*, *User Changes* and *Workflow Server* web applications.

Table 4.8. Properties for persistence of collaboration data to MongoDB

MongoDB Authentication

MongoDB authentication is enabled on deployment level and the user coremedia/coremedia is created by default.

Authentication is performed against the admin database. Example:

```
use admin
db.auth('coremedia','coremedia')
```

The default `mongodb.client-uri` is configured with credentials, for example

```
mongodb.client-uri=mongodb://coremedia:coremedia@${installation.host}:27017
```

For development with a MongoDB without authentication, either remove the credentials prefix from the `mongodb.client-uri` property or create a user with:

```
use admin
db.createUser({user: 'coremedia', pwd: 'coremedia', roles:
['userAdminAnyDatabase', 'dbAdminAnyDatabase', 'readWriteAnyDatabase']});
```

4.5.3 Recovery of Collaborative Components Database

In this chapter you will get to know how to back up and recover the database, deployed with CoreMedia's collaborative components.

4.5.3.1 Backup Strategy

You need to have database backups to recover from database failures. The backups are created with database tools. The exact backup procedure depends on your database product and likely on the configuration of your database. The chronological order of the backups is crucial:

1. Backup the CoreMedia editorial comments database.
2. Backup the CoreMedia collaborative components database.
3. Backup the *Content Management Server*'s database.

CAUTION

Note, that recovery will work correctly, if this given chronological order of backups is respected. The content of the *Content Management Server* must be newer than the content of the collaborative components database. The time between the single backups should be short.

See Content Server Manual for information how to back up the Content Server's database.

You can find an overview about backup of MongoDB and possible backup strategies here.

4.5.3.2 Recovery of the Collaborative Components Database

In order to recover the database of the collaborative components, proceed as follows:

1. Stop *CoreMedia Studio*, *Workflow Server* and *User Changes*.
2. Stop the *Content Management Server*. The sessions of the connected clients will be closed and no more content changes are possible.
3. Restore the *Content Management Server* with a backup. Note, that this backup must be newer than the backup of the collaborative components database.
4. Restart the *Content Management Server*.
5. Recover the database of *CoreMedia's collaborative components*.
6. Restart *CoreMedia Studio*, *Workflow Server* and *User Changes*.

4.6 CoreMedia Licenses

CoreMedia CMS uses file based licenses. Only server applications (*Content Management Server* and *Live Servers*) have a license file on their own. All other applications are licensed by the license file of the server they connect to. The license file will be read in from the directory defined in the property `cap.server.license` and will be validated each time the licensed application starts. If the license is valid, the application will start properly. CoreMedia distinguishes between two kinds of licenses:

- *Time-based license*
  Limits the use of an application to a specific period.
- *IP-based license*
  Limits the use of an application to a specific computer, defined by its IP address and/or host name.

Both license types can define a valid *CoreMedia CMS* release using the `release` attribute. If you use time-based licenses, the application will not start if the license has expired. In addition, the license file defines a grace period. You receive a notification, after exceeding the grace period. You will see this warning each time you start the *Site Manager* and in the log files of the application.

Both license types may limit the number of clients which can connect to the application simultaneously. This is achieved, using the following concepts:

- *Named user*
  A named user is a specific *CoreMedia CMS* user, known by the system. Each service connects as a user to the server. The attribute `named-users` defines the maximum number of users which are allowed to use a specific service.
- *Concurrent user*
  Concurrent users are users which are connected simultaneously to the server. The attribute `concurrent-users` defines the maximum number of named users which are allowed to connect simultaneously.
- *Multiplicity*
  A named user may connect several times to the server (start two site managers, for example). The attribute `multiplicity` defines the maximum number of allowed connections for a named user.

Use the utility `sessions` (see Section 3.13.1.8, “Sessions” in *Content Server Manual*) to get this information and the utility `usedlicenses` (see Section 3.13.2.21, “Usedlicenses” in *Content Server Manual*) to free used licenses. If the built-in user `admin` (user ID 0) has no open sessions, that user may log in to the Content Server even if the licenses are otherwise exhausted. This makes it possible to start the utilities for recovering from a license shortage in any case.

A server license can be exchanged at runtime without restarting the server. The property `cap.server.license` defines the location of the license file. When the file or location changes, the server will automatically reload the license. Reloading the license will not cause any open sessions to be closed, even if the new license is more restrictive than the old one.

Example:

```
<LicenseConfiguration>
  <Server type="production"/>
  <Property name="licensed-to" value="Customer"/>
  <Property name="workflow" value="enabled"/>
  <Property name="elastic-social" value="enabled"/>
  <Property name="personalization" value="enabled"/>
  <Property name="analytics" value="enabled"/>
  <Property name="livecontext" value="enabled"/>
  <Property name="brand-blueprint" value="enabled"/>
  <Property name="asset-management" value="enabled"/>
  <Property name="id" value="10394"/>
  <Valid from="01.01.2015" until="01.12.2015" grace="01.11.2015"/>
  <License service="editor" concurrent-users="30000"
   named-users="200"/>
  <License service="system" concurrent-users="5"
   named-users="25"/>
  <License service="webserver" concurrent-users="15"
   named-users="50"/>
  <License service="studio" concurrent-users="15"
   named-users="50"/>
  <License service="workflow" concurrent-users="600"
   named-users="200"/>
  <License service="importer" concurrent-users="2"
   named-users="25"/>
  <License service="publisher" concurrent-users="33"
   named-users="200"/>
  <License service="debug" concurrent-users="100"
   named-users="100"/>
  <License service="replicator" concurrent-users="5"
   named-users="10"/>
  <License service="feeder" concurrent-users="2"
   named-users="10"/>
</LicenseConfiguration>
```

Example 4.2. A sample license file

The attributes of the License file elements have the following meaning:

Element	Attribute	Description
Server	type	The type of the server for which the license is valid. Possible values are: • *production*: The *Content Management Server* • *publication*: The Master Live Server • *live*: The *Replication Live Server*

Element	Attribute	Description
Property	name	The aim of the property. Possible values are: • licensed-to: The customer to which the system is licensed. • workflow: Defines if the programmable workflow is licensed ["enabled"]. • analytics: Defines if Analytics is licensed ["enabled"]. • elastic-social: Defines if Elastic Social is licensed ["enabled"]. • personalization: Defines if Adaptive Personalization is licensed ["enabled"]. • elastic-social: Defines if Elastic Social is licensed ["enabled"]. • id: The unique ID of the license.
	`value`	The value of the property. The possible values depend on the name attribute.
Valid	`from`	The starting date of the validity of the license.
	`until`	The end date of the validity of the license.
	`grace`	The starting point of the grace period.
	`release`	The CoreMedia release for which the license is valid.
	`host`	The host name for which the license is valid.
	`ip`	The IP address for which the license is valid.
License	`service`	The name of a service which might connect to the server.
	`concurrent-users`	The maximum number of simultaneously allowed sessions of this service.
	`named-users`	The maximum number of users which are allowed to be allocated to the service.
	`multiplicity`	The maximum number of sessions a user is allowed to open.

Table 4.9. Elements of a license file

4.7 Logging

An important element in the monitoring of *CoreMedia CMS* applications is logging. Without recording relevant information of the system it is often impossible to find out when an irregularity occurred.

Logback

CoreMedia Content Cloud uses Logback for logging. You can use all features of Logback when configuring the log configuration of CoreMedia applications. See Logback documentation for details http://logback.qos.ch/documentation.html. One exception is Apache Solr, which uses Apache Log4J.

4.7.1 Logging Configuration for Applications

CoreMedia applications use Logback. The log configuration for each application is packaged into each application `jar` archive. To configure the log level of a specific logger, you only need to set an application property, which follows the Spring Boot standard for log configuration. You can set the property in any of the location described here.

If for example you want to set the log level of the `com.coremedia` logger to `debug`, set the following property and restart the application.

```
logging.level.com.coremedia=debug
```

If you want to change the log level at runtime without restart, you can use the logger management actuator for an application if enabled. If that is the case, you can use a simple PUT request to set the new level. Please visit the official Spring Documentation for more details.

If you want to change other logging characteristics, you need to add a `src/main/resources/logback-spring.xml` file in each Spring Boot application module before building it.

4.7.2 Logging Configuration for Apache Solr

Apache Solr uses Apache Log4J 2 as log framework, which is configured in the file `server/resources/log4j2.xml` in the Solr installation.

Note that you can use the Solr admin page to view log messages and change the log level at runtime. Alternatively you could configure Apache Solr to use Logback as well, but then you cannot use the logging functionality of the Solr admin page. See Solr Reference Guide: Configuring Logging for details on Solr log configuration.

4.7.3 Logging Configuration for Command-Line Tools

The logging configuration for each command-line tool is taken from the `tools-logback.xml` file in `properties/corem` directory by default. You can use a customized configuration file and add the file name to the system properties when initializing the application with:

```
-Dlogback.configurationFile=file://localhost/<PathtoYourFile>/<yourFileName>.xml
```

You will find the default logging facilities of CoreMedia applications in the default logging configuration.

`stdout/stderr` Output

Enter the location for the `stdout/stderr` output of an application and any other third-party program in the corresponding JPIF start file of the application. To do so, configure the parameter `OUTPUT_REDIRECT` in the corresponding JPIF file of the application as it is described in this file.

4.8 Security

To secure a *CoreMedia CMS* installation against unauthorized access, you have to consider the various system components, their operating environment and their interconnection.

As a general rule, protect the system on all possible levels. For example, good passwords and a good network infrastructure complement each other, but do not make each other obsolete.

4.8.1 Overall Deployment

Typically, a firewall is in place between the content management environment and the delivery environment, limiting the information flow from the untrusted Internet environment. Additionally, a firewall in front of the delivery environment may further reduce the number of exposed system components and communication ports of the delivery environment.

Typically, access from the Internet is granted to a load balancer, only, which delegates requests to the CAEs.

Especially services that are not properly protected by authentication must never be exposed outside of the local network. Examples for this rule would be a MongoDB in its default configuration or a Solr instance. For more details how to secure Solr in the *CMCC* context, see Section 4.8.7.1, “Securing the Solr Search Engine” [55].

If necessary, access to the content management environment may be granted through a VPN, allowing remote editor connections.

Much of the communication between system components happens through either HTTP or CORBA. You can find details and helpful security hints in Section 4.4, “Communication between the System Applications” [30]. In particular it is shown how CORBA can be layered on top of SSL.

4.8.2 Open Ports

CoreMedia components communicate through various TCP based protocols. To that end, server ports are opened. You should make sure that only required ports are open.

The application server can open multiple connectors, for example, supporting both HTTP and AJP. You should disable the ports you don't need.

Prefer HTTPS over HTTP and, where possible, disable the HTTP ports entirely. See Section 4.4.5, "Preparing Spring Boot applications for HTTPS Connection" [40] for instructions on the Tomcat configuration.

Both *Content Server* and *Workflow Server* need a CORBA server port opened by the ORB. They can use a dedicated ORB, but typically they use the ORB provided by the application container as described in Section 4.4, "Communication between the System Applications" [30].

CORBA clients will also instantiate an ORB if it is not provided by an application container.

Server ports that listen to many network interfaces are more prone to attacks. In Section 4.4.3, "Binding Only a Single Network Interface" [34] you can find procedures to limit the number of network interfaces bound when providing services.

Services can be managed by means of JMX. Use the existing JMX connectors and do not open additional connectors. Make sure that accesses to the connectors are subject to authentication.

4.8.3 Passwords

Change all standard passwords of built-in users immediately after installation. Use good passwords.

When providing a password to command line tools in automated procedures, prefer the environment variable `REPOSITORY_PASSWORD` to the `-p` command line argument. If possible, retrieve the password immediately before calling the command line tool from a secure password vault. Make sure that the environment variable does not remain set for too long.

The users' passwords are stored by the *Content Servers* as salted hashes. The hash algorithm can be configured using the server property `cap.server.login.passwordHashAlgorithm`, which should be set to `bcrypt:N` where N is the load factory of the bcrypt password hashing algorithm. Higher values of N slow down the hashing performance and improve security. Set N to at least 10 and choose higher values if the CPU performance allows it.

The passwords can be encrypted additionally by using the tool `cm encryptpasswords` as described in Section 3.13.2.7, "Encryptpasswords" in *Content Server Manual*.

Some passwords stored in configuration files can be encrypted using the tool `encryptpasswordproperty` as described in Section 3.13.1, "Information" in *Content Server Manual*. This applies to:

- database passwords used by *Content Server*, *Workflow Server* and *Studio Server*
- passwords for connecting to *Content Server* and *Workflow Server*,

- passphrases for the CORBA-over-SSL keystore.

Passwords for connecting to an LDAP server, to a MongoDB or to a Solr cannot be protected in the same manner.

4.8.4 URL Injection

Blobs can be stored as URLs that are resolved when the blob is accessed (persistent URL blobs). This feature is restricted to HTTP and HTTPS URLs by default, because other URLs like file URLs might point to sensitive data that can be exfiltrated by injecting a malicious URL into the content repository. To control the allowed URLs for URL blobs, set the *Content Server* property `cap.server.blobUrlPattern` to a regular expression that matches the allowed URLs. Note that the pattern is used to check URLs during writes and does not affect already stored blobs.

4.8.5 Data Storage

Make sure that read and write rights for databases and for file systems containing CMS installations and data are reduced to a minimum.

Some CoreMedia components are configured to write heap dumps when they run out of memory, helping you to quickly diagnose critical failures. Make sure that the directories to which these heap dumps are written are properly secured, because heap dumps contain sensitive information like passwords, which might not have been disposed by the garbage collector.

Log files, too, must only be readable by authorized staff. They can contain hints that help an attacker spot weaknesses.

The temporary directory of Java as configured by the system property `java.io.tmpdir` is used for some data. Often it points to a directory that is writable by everyone. Preferably, you should use a secured and isolated temporary directory for each component. Alternatively you can configure the storage directory paths explicitly as far as they default to the temporary directory.

On some operation systems, `java.io.tmpdir` is mapped to a directory that is regularly cleaned up by the operation system. For short running processes this behaviour won't affect the application, but for long running processes, this may result in unintended cache data loss and application faults. To prevent this, you should always configure cache locations such as the UAPI blob cache to a different directory outside of these automatically cleaned paths.

The most important data storage locations are summarized in the following list:

- the databases of all *Content Servers* and the *Workflow Server*,
- if so configured, the blob stores of all *Content Servers*,
- the stores of all MongoDB instances,
- the input directories of importer processes,
- the Solr home directory, which should be created before Solr is started so that it is does not default to the Java temporary directory,
- if so configured, the serialization file of the Control Room in-memory store,
- the temporary file stores of all *Replication Content Servers*, as configured in the `replicator.tmpDir` property,
- the blob caches of all *Unified API* connections as configured in the `repository.blobCachePath` property or the `Cap.BLOB_CACHE_PATH` connection attribute (defaulting to the Java temporary directory),
- the Java temporary directory of *Site Manager* instances,
- the installation directories of all components,
- the logging directories.

4.8.6 Content Delivery

The most visible service of a CoreMedia CMS installation are the delivery CAEs.

Validate request URLs and request parameters. Make sure a properly styled, but terse error page is in place to avoid giving hints about the cause of malfunctions. Make sure to escape text data properly to avoid cross-site scripting attacks.

4.8.7 Third-party Software

Make sure to apply security patches to the operating systems, the Java installation, the databases and all other third-party software. Refer to the supported environments documentation for details on the tested versions of all third-party software.

CoreMedia Studio and some other components run in web browsers. Make sure to update the browsers regularly to the latest version. Being manually operated, browsers offer a particularly large attack surface.

4.8.7.1 Securing the Solr Search Engine

The Solr engine is no public service within the *CMCC* architecture. Therefore, any external requests should be blocked by a firewall.

Index update requests (like "delete all") from internal computers should be restricted by Basic Authentication. For details, see Solr Reference Guide: Securing Solr. All related *CMCC* components are capable of Solr Basic Authentication and feature the configuration properties `solr.username` and `solr.password` to set the credentials.

4.8.8 Customizations

Both frontend and backend applications are typically deployed with code fragments for customization or may in some cases be written from scratch based on the CoreMedia APIs. Make sure to review source code for security issues.

Validate input data and handle imported data robustly. Be careful when external data causes the access of local resources, for example reading files or content objects or making server-side remote requests. XML parsing may leak local data through XML external entity (XXE) references. While the XML API in `com.coremedia.xml` has been hardened as far as possible, the native Java XML parsing might be more vulnerable.

Java serialization and deserialization must be used with care, because the JVM suspends certain protection mechanism for these operations, allowing both data leaks and code execution.

Note that a *Unified API* connection can perform all operations for which its logged-in user is authorized. A *Unified API* connection for the users `studio` and `workflow` may even incorporate other users, thereby gaining full access. This means that extensions of the *Workflow Server*, and *Studio* must be particularly well checked.

4.9 JMX Management

By default, all CoreMedia applications register relevant resources via JMX as MBeans for management and monitoring purposes. This might range from simple log configuration up to repository statistics or cache capacities. You will find a list of the functionality supported via JMX in most of the CoreMedia application manuals.

All resources are registered using Spring's ability to register and export MBeans to an MBean server. You can access the MBean server with any JMX client without configuration, if this client is running on the same machine. A common JMX client is JConsole, which is bundled with Oracle's JDK but you can also choose one of the freely available clients.

4.10 Actuator Endpoints

Spring Boot Actuator is a part of the Spring Boot framework to provide production ready features for all applications to integrate them into your production landscape.

CoreMedia enables a set of these endpoints by default and adds custom endpoints on top to provide a seamless integration for your operational needs. This section will focus only on the customizations and additions to the default actuator set.

Some of the added endpoints will apply to multiple applications and will be activated by application properties and Spring autoconfigurations and some are only available in specific applications and need to be activated on demand.

4.10.1 Info Endpoint

The info endpoint exposes arbitrary application info. For CoreMedia applications, it exposes build information and the servlet container's name and version. If the `dev` or `local` spring profile is active, it also exposes basic information about the application's dependencies.

The info endpoint can be enriched with custom data by adding beans implementing `org.springframework.boot.actuate.info.InfoContributor`.

4.10.2 Health Endpoint

The health endpoint can be enriched with custom health checks by adding classes extending the `org.springframework.boot.actuate.health.HealthIndicator` class of the Spring Boot framework. Each health indicator is a bean with a name suffix `healthIndicator`. In the example below we will describe the usage on a health indicator with the name `uapiConnection`.

By default the health indicator can be enabled or disbled by configuring the following property:

Endpoint properties:

```
management.health.uapiConnection.enabled=true
```

When activated, the health indicator will extend the default health endpoint with its name as its subpath.

Endpoint details activated:

```
management.endpoint.health.show-details=always
```

Endpoint URL:

```
http://localhost:8081/actuator/health/uapiConnection
```

When requested using a GET HTTP request, the endpoint will response with a HTTP return code, matching the state of the check.

Response:

```
{
  "status": "UP", ❶
  "uapiConnection": {
    "status": "UP", ❷
    "details": {
      "content repository": "OK" ❸
    }
  }
}
```

❶ Global health endpoint status.
❷ Health status of the `uapiConnection` health indicator.
❸ Details of the `uapiConnection` health indicator when detailed view is enabled.

4.10.2.1 CoreMedia Health Indicator

id	Description
uapiConnection	Checks the state of the UAPI connection. In the details view, the status of each connected repository is shown (content repository, workflow repository).
uapiConnection-Readiness	Checks also the future state of the UAPI connection based on runlevel changes of the repository.
runlevel	Checks the runlevel of a content server. See extra description how to use this in Kubernetes.
mongoDb	Checks the health of the MongoDB connection if available.
elasticSolr	Checks the health of the Solr connection in the elastic social context if available.
contentSolr	Checks the health of the Solr connection for content search if available.

id	Description
blob-CacheDiskSpace	Checks if enough diskspace is available for the UAPI blobcache. The threshold can be configured using the property `management.health.blobCacheDiskspace.threshold`.
transformedBlob-CacheDiskSpace	Checks if enough diskspace is available for the transformed blobcache. The threshold can be configured using the property `management.health.transformedBlobCacheDiskspace.threshold`.
editorialCom-mentsDatasource	Checks the state of the editorial comments datasource.
commerceEnd-pointHealth	Checks the health of the commerce endpoint. In the details the health for each connected commerce system adapter is shown.
replicator	Checks the state of the replication process. In the details, it is possible to track the amount of unreplicated events. With the property `management.health.replicator.uncompleted-events-threshold`, it is possible to define a threshold to fail the check.
contentFeeder-HealthIndicator	Checks the state of the content feeder.

Table 4.10. Health indicators

4.10.2.2 Health Endpoints in the Context of a Kubernetes Deployment

In a Kubernetes deployment, you have three different probing mechanism to determine the state of your pod.

- a startupProbe to delay adding the pod to the loadbalancing until it is ready for business. This probe is especially relevant if an application takes a long initialization time aside from being ready.
- a readinessProbe to signal the loadbalancing that the application pod is ready to accept traffic.
- a livenessProbe to signal that the application pod is alive and should not be removed for rescheduling.

In order to use these probes, there are dedicated endpoints below the health endpoint:

- `container:8081/actuator/health/readiness`
- `container:8081/actuator/health/liveness`

These endpoints are activated by the following Spring Boot properties, which are enabled by default in the Blueprint workspace:

```
management.endpoint.health.probes.enabled=true
management.health.livenessstate.enabled=true
management.health.readinessstate.enabled=true
```

In UAPI clients, the `uapiConnectionReadiness` health indicator can be added to the readiness health endpoint to signal Kubernetes an upcoming runlevel change of the content-server to remove the pod from loadbalancing and deny further traffic. The runlevel switch can then be triggered using the Section 4.10.7, "Content Server Runlevel Endpoint" [74] with a grace period in the content-server pod during a `preStop` lifecycle hook followed by a sleep timeout equal to the grace period. This way, the grace period and the sleep timeout are defining the connection draining period of the client.

To include the `uapiConnectionReadiness` health endpoint in the readiness endpoint, you need to set the following Spring Boot properties:

```
management.health.uapiConnectionReadiness.enabled=true
management.endpoint.health.group.readiness.include=readinessState,uapiConnectionReadiness
```

4.10.3 Cache Endpoint

The `cache` endpoint provides the possibility to get information about the configuration and usage of the CoreMedia cache, and to change its capacity, clear the cache, or trigger a cache eviction. For more information about the cache and its properties, see the API documentation of Java class `com.coremedia.cache.Cache`.

Endpoint properties:

```
management.endpoint.cache.enabled=true
```

Endpoint URL:

```
http://localhost:8081/actuator/cache
```

4.10.3.1 Retrieving Cache Classes

To retrieve all cache classes, make a `GET` request to `/actuator/cache`:

```
curl http://localhost:8081/actuator/cache
```

The JSON response body lists cache classes with their capacity and current level, as in the following example:

Response:

```
{
  "cacheClasses": {
    "ALWAYS_STAY_IN_CACHE": {
      "capacity": 9223372036854775807,
      "level": 26,
      "href": "http://localhost:8081/actuator/cache/ALWAYS_STAY_IN_CACHE"
    },
    "DIGEST": {
      "capacity": 9223372036854775807,
      "level": 0,
      "href": "http://localhost:8081/actuator/cache/DIGEST"
    },
    "com.coremedia.cap.disk": {
      "capacity": 10737418240,
      "level": 0,
      "href": "http://localhost:8081/actuator/cache/com.coremedia.cap.disk"
    },
    "com.coremedia.cap.heap": {
      "capacity": 104857600,
      "level": 88905639,
      "href": "http://localhost:8081/actuator/cache/com.coremedia.cap.heap"
    },
    "java.lang.Object": {
      "capacity": 10000,
      "level": 107,
      "href": "http://localhost:8081/actuator/cache/java.lang.Object"
    }
  }
}
```

The cache classes `ALWAYS_STAY_IN_CACHE` and `DIGEST` are predefined classes with unlimited capacity. Cache class `com.coremedia.cap.heap` is used for internal caching in the Unified API, with capacity and level being rough estimates of the required heap memory in byte. Other cache classes may use different units for capacity and level, and many simply count the number of cached values like the cache class `java.lang.Object`, which is configured to hold up to 10,000 objects in this example.

The `"href"` links in the JSON response can be used to get information about a single cache class. These links are only present if the actuator endpoint is invoked over HTTP. Like many other endpoints, the `cache` endpoint may also be exposed and invoked over JMX, in which case the response will not contain such links.

To retrieve the capacity and level of a single cache class, make a `GET` request to `/actuator/cache/<cacheClass>`, for example:

```
curl http://localhost:8081/actuator/cache/com.coremedia.cap.heap
```

The response will look like

Response:

```
{
  "capacity": 104857600,
  "level": 88905639
}
```

If the specified cache class is neither configured nor used, then the request will be answered with HTTP status code 404 (Not Found).

4.10.3.2 Retrieving CacheKey Classes

For each cache class, different `com.coremedia.cache.CacheKey` classes may be used to evaluate and get cached values. You can use the `keys` query parameter to retrieve all used `CacheKey` classes for some cache class by making a `GET` request to `/actuator/cache/<cacheClass>?keys=true` as in the following example.

Note, that the endpoint implementation has to scan all cache entries to find the `CacheKey` classes. For large caches, this can be expensive, because the cache is temporarily locked against updates.

```
curl http://localhost:8081/actuator/cache/com.coremedia.cap.heap?keys=true
```

The response lists used `CacheKey` classes with the number of cache entries and the level indicating how much space they occupy of the cache class capacity. For example, a response may start like this:

Response:

```
{
  "capacity": 104857600,
  "level": 88905639,
  "keys": {
    "com.coremedia.cap.undoc.multisite.impl.VariantsCacheKey": {
      "count": 6,
      "level": 6364,
      "href":
"http://localhost:8081/actuator/cache/com.coremedia.cap.heap/com.coremedia.cap.undoc.multisite.impl.VariantsCacheKey"

    },
    "com.coremedia.cotopaxi.content.ChildrenKey": {
      "count": 983,
      "level": 493484,
      "href":
"http://localhost:8081/actuator/cache/com.coremedia.cap.heap/com.coremedia.cotopaxi.content.ChildrenKey"

    }
```

The `"href"` links in the JSON response can be used to get information about a single `CacheKey` class. These links are only present if the actuator endpoint is invoked over HTTP. Like many other endpoints, the `cache` endpoint may also be exposed and invoked over JMX, in which case the response will not contain such links.

To retrieve the count and level of a single `CacheKey` class in a cache class, make a `GET` request to `/actuator/cache/<cacheClass>/<CacheKey>`, for example:

```
curl
http://localhost:8081/actuator/cache/com.coremedia.cap.heap/com.coremedia.cotopaxi.content.ChildrenKey
```

The response will look like

Response:

```
{
  "count": 983,
  "level": 493484
}
```

4.10.3.3 Browsing Cache Entries

To view cache entries for a cache class or `CacheKey`, use the endpoint's `entries` query parameter by making a `GET` request to `/actuator/cache/<cacheClass>?entries=true` or `/actuator/cache/<cacheClass>/<CacheKey>?entries=true`.

Note, that the endpoint implementation has to scan all cache entries to find matching entries. For large caches, this can be expensive, because the cache is temporarily locked against updates.

The following example requests entries for content properties that are cached internally by the *Unified API*:

```
curl
http://localhost:8081/actuator/cache/com.coremedia.cap.heap/com.coremedia.cotopaxi.content.PropertiesKey?entries=true
```

The response body starts like this:

Response:

```
{
  "count": 3327,
  "level": 27444160,
  "entries": {
    "total": 3327,
    "offset": 0,
    "limit": 10,
    "elements": [
      {
        "keyClass": "com.coremedia.cotopaxi.content.PropertiesKey",
        "keyToString": "key.properties(448)",
        "weight": 432696,
        "dependencies": {
          "total": 1,
          "offset": 0,
          "limit": 10,
          "elements": [
            {
```

```
            "class": "com.coremedia.cotopaxi.content.PropertiesDependency",
            "toString": "dependency.properties(coremedia:///cap/content/448)"
          }
        ]
      },
      "dependents": {
        "total": 0,
        "offset": 0,
        "limit": 10,
        "elements": []
      }
    },
    {
      "keyClass": "com.coremedia.cotopaxi.content.PropertiesKey",
      "keyToString": "key.properties(202)",
      "weight": 80150,
...
```

The actual response is longer and shows the first 10 cache entries of 3327 in total, sorted by descending weight, so that the entry that occupies the most space in the cache comes first. Each entry is listed with the name of its `CacheKey` class (`keyClass`), the `#toString` representation of the `CacheKey` object (`keyToString`), the space it occupies in the cache (`weight`), its `dependencies` and `dependents`.

You can control the output with additional optional query parameters:

Query Parameter	Default	Description
values	`false`	Set to `true` to include the class and `toString` representation of cached values in the response.
entriesSort	`weight desc`	Sort criteria for returned entries. Possible values are `weight`, `keyClass`, `keyToString`, `dependencies` (number of) and `dependents` (number of). The sort direction can be specified by appending a space character followed by `asc` for ascending or `desc` for descending sort. The default sort direction is ascending. Multi-level sorting can be specified by comma-separated values, for example url-encoded query parameter `entriesSort=keyClass+asc,weight+desc` would sort by ascending `CacheKey` class, while equal key classes would be sorted by descending weight.
entriesOffset	0	The number of cache entries to skip.
entriesLimit	10	The maximum number of cache entries to return.

Query Parameter	Default	Description
dependenciesOffset	0	For each cache entry, the number of dependencies to skip.
dependenciesLimit	10	For each cache entry, the maximum number of dependencies to return.
dependentsOffset	0	For each cache entry, the number of dependents to skip.
dependentsLimit	10	For each cache entry, the maximum number of dependents to return.
stringLimit	200	For the string representation of keys, dependencies and values, the maximum length before strings get shortened in the response. Set to 0 to not return string representations.

Note, that the endpoint calls `CacheKey#toString`, the `#toString` method of dependency objects, and if `values=true`, the `#toString` method of cached values. Depending on the implementation of `#toString` methods, these can be expensive operations. You can avoid that `#toString` methods are called by setting query parameter `stringLimit` to `0` and not using the `keyToString` sort criteria.

4.10.3.4 Set Cache Capacity

To change the capacity for a certain cache class, make a `POST` request to `/actuator/cache/<cacheClass>` with a JSON body that specifies the capacity for the cache class, as shown in the following example:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"capacity": 20000}' \
     http://localhost:8081/actuator/cache/java.lang.Object
```

The response includes a message about the performed change, and the new state of the cache class:

Response:

```
{
  "message": "Capacity changed for cache class 'java.lang.Object': 10000 ->
 20000",
  "result": {
    "capacity": 20000,
    "level": 107
  }
}
```

Note, that setting a smaller capacity for a cache class does not necessarily lead to an immediate eviction of cached values. If you want to reduce the current cache level with

a reduced capacity, you can either make a separate request to trigger an eviction as described in Section 4.10.3.6, "Trigger Cache Eviction" [67] or make a `POST` request with a JSON body to set both a new capacity and trigger an eviction:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"capacity": 100, "evict": true}' \
     http://localhost:8081/actuator/cache/java.lang.Object
```

The response includes a message about the performed changes, and the new state of the cache class:

Response:

```
{
  "message": "Capacity changed for cache class 'java.lang.Object': 20000 ->
 100; Cache eviction triggered for cache class: 'java.lang.Object'.",
  "result": {
    "capacity": 100,
    "level": 90
  }
}
```

4.10.3.5 Clear the Cache

To clear the cache and remove all cached entries, make a `POST` request to `/actuator/cache` with a JSON body as in the following example:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"clear": true}' \
     http://localhost:8081/actuator/cache
```

The response includes a message stating that the cache was cleared, and the state of the cache afterwards, as it would be returned by a `GET` request to the same URL. The following example just shows the start of the response body:

Response:

```
{
  "message": "Cache cleared",
  "result": {
    "cacheClasses": {
      "ALWAYS_STAY_IN_CACHE": {
        "capacity": 9223372036854775807,
        "level": 0,
        "href": "http://localhost:8081/actuator/cache/ALWAYS_STAY_IN_CACHE"
      },
...
```

To clear the cache for a single cache class only, a similar `POST` request can be made to `/actuator/cache/<cacheClass>` as in the following example:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"clear": true}' \
     http://localhost:8081/actuator/cache/java.lang.Object
```

Again, the response includes a message about the operation and the state of the cache class afterwards:

Response:

```
{
  "message": "Cache cleared for cache class 'java.lang.Object'.",
  "result": {
    "capacity": 10000,
    "level": 0
  }
}
```

4.10.3.6 Trigger Cache Eviction

To trigger a cache eviction, make a `POST` request to `/actuator/cache` with a JSON body as in the following example:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"evict": true}' \
     http://localhost:8081/actuator/cache
```

The response includes a message stating for which cache classes an eviction was triggered, and the state of the cache afterwards, as it would be returned by a `GET` request to the same URL. The following example just shows the start of the response body:

Response:

```
{
  "message": "Cache eviction triggered for cache classes:
[com.coremedia.cap.disk, ...",
  "result": {
...
```

To trigger a cache eviction for a single cache class only, a similar `POST` request can be made to `/actuator/cache/<cacheClass>` as in the following example:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"evict": true}' \
     http://localhost:8081/actuator/cache/java.lang.Object
```

Again, the response includes a message about the operation and the state of the cache class afterwards:

Response:

```
{
  "message": "Cache eviction triggered for cache class: 'java.lang.Object'.",
```

```
  "result": {
    "capacity": 10000,
    "level": 107
  }
}
```

4.10.4 CapConnection Endpoint

The capconnection endpoint exposes information about the state of the cap connection in client applications.

Endpoint properties:

```
management.endpoint.capconnection.enabled=true
```

Endpoint URL:

```
http://localhost:8081/actuator/capconnection
```

When requested with a GET request, the endpoint responds with the state of the cap connection.

Response:

```
{
  "url": "http://localhost:8080/ior",
  "user": {
    "domain": "",
    "name": "webserver"
  },
  "state": {
    "disrupted": false,
    "numberOfSUSessions": 0,
    "open": true,
    "stable": true
  },
  "content-repository": {
    "available": true,
    "healthy": true,
    "required": true
  },
  "workflow-repository": { ... },
  "caplist-repository": { ... },
  "events": {
    "timeSinceLastEventRetrievalMS": 54147,
    "latestReceivedContentEventSequenceNumber": 112678,
    "eventRetrievalDelayMS": 60000,
    "latestContentEventSequenceNumber": 112678,
    "eventChunkSize": 1000
  },
  "heap-cache": {
    "level": 2495694,
    "faults": 134,
    "size": 104857600
  },
```

```
  "blob-cache": { ... }
}
```

With a POST request to the endpoint, you can change some of the properties.

Configure cap connection:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"blobStreamingThreads": 3, "eventChunkSize": 1000}' \
     http://localhost:8081/actuator/capconnection
```

The configurable properties are

- `blobCacheSize`
- `blobStreamingSizeThreshold`
- `blobStreamingThreads`
- `eventChunkSize`
- `heapCacheSize`
- `maxCachedBlobSize`

For the meanings of the properties see the API documentation of `com.coremedia.cap.common.CapConnectionManager`.

4.10.5 Customizations Endpoint

The customizations endpoint exposes the CoreMedia spring bean customizations report as XML.

4.10.6 Metrics Endpoint

The metrics endpoint is a standard Spring Boot actuator endpoint that you can use to examine metrics collected by the application. In addition to standard Spring Boot metrics, some CoreMedia applications export additional metrics that are described in this chapter.

4.10.6.1 Cache Metrics

There are several metrics that collect information about application caches. Access to them is divided to

- CoreMedia-based cache and

- other caches.

CoreMedia Cache

The CoreMedia cache (based on `com.coremedia.cache.Cache`) provides more detail and uses a different set of `meter names`. Metrics are collected per cache class, which can be selected with the `class` tag.

For example, you can request the current cache level for cache class `com.coremedia.cap.heap` with

```
http://localhost:8081/actuator/metrics/coremedia.cache.level
  ?tag=class:com.coremedia.cap.heap
```

The response will then provide the value for the cache level:

```
{
  "name": "coremedia.cache.level",
  "description": "The total weight of all values which are currently held in
 the cache",
  "baseUnit": null,
  "measurements": [{
      "statistic": "VALUE",
      "value": 1327946.0
  }],
  "availableTags": []
}
```

The following table lists available metrics for the CoreMedia cache. Some metrics support additional tags to further drill down into measurements. The tag `class` is available for all metrics to set cache class.

Meter Name	Tags	Description
coremedia.cache.capacity		The configured capacity.
coremedia.cache.level		The total weight of all values currently held in the cache.
coremedia.cache.size		The number of entries in this cache.
coremedia.cache.gets	result:hit	The number of cache hits, which are cache lookups that returned a cached value.
coremedia.cache.gets	result:miss	The number of cache misses, which are cache lookups that had to evaluate a value.
coremedia.cache.puts		The number of values inserted into the cache, after successful evaluation or injection.

Meter Name	Tags	Description
coremedia.cache.evictions		The number of values evicted from the cache.
coremedia.cache.updates		The number of values updated after re-computation.
coremedia.cache.removals		The number of values removed from the cache after invalidation.
coremedia.cache.evaluation.duration		The time the cache has spent evaluating values.
coremedia.cache.eviction.duration		The time the cache has spent evicting values.

Table 4.11. CoreMedia Cache Metrics

Other Caches

On some CoreMedia applications, other caches are employed in addition to the CoreMedia cache.

For example, you can request the current cache size for cache `richtext` on a *Headless Server* with

```
http://localhost:8081/actuator/metrics/cache.size?tag=name:richtext
```

The response will then provide the value for the cache size:

```
{
  "name": "cache.size",
  "description": "The number of entries in this cache. This may be an
approximation, depending on the type of cache.",
  "baseUnit": null,
  "measurements": [{
      "statistic": "VALUE",
      "value": 91.0
  }],
  "availableTags": [
    {
      "tag": "name",
      "values": [
        "richtext"
      ]
    },
    {
      "tag": "cacheManager",
      "values": [
```

```
        "cacheManager"
      ]
    }
  ]
}
```

Depending on the application, different `cache names` are available. The list of all caches, if any are present, can be retrieved with a call to

```
http://localhost:8081/actuator/metrics/cache.size
```

If no other caches are present for the CoreMedia application, the request will return code 404.

The following table lists available metrics for other caches which are present on some CoreMedia applications. Some metrics support additional tags to further drill down into measurements.

Meter Name	Tags	Description
cache.size		The number of entries in this cache.
cache.gets	result:hit	The number of cache hits, which are cache lookups that returned a cached value.
cache.gets	result:miss	The number of cache misses, which are cache lookups that had to evaluate a value.
cache.puts		The number of values inserted into the cache, after successful evaluation or injection.
cache.evictions		The number of values evicted from the cache.

Table 4.12. Other Cache Metrics

4.10.6.2 CapConnection Metrics

The client components provide the following metrics about their cap connections:

- `coremedia.connection.blobcachefaults`
- `coremedia.connection.contentrepositoryavailable`
- `coremedia.connection.disrupted`
- `coremedia.connection.eventretrievaldelay`

- `coremedia.connection.heapcachefaults`
- `coremedia.connection.latestcontenteventsequencenumber`
- `coremedia.connection.latestreceivedcontenteventsequencenumber`
- `coremedia.connection.numberofsusessions`
- `coremedia.connection.open`
- `coremedia.connection.stable`
- `coremedia.connection.timesincelasteventretrieval`
- `coremedia.connection.workflowrepositoryavailable`

For the meanings of the properties see the API documentation of `com.coremedia.cap.common.CapConnectionManager`. Boolean values are represented by the numbers 0.0 and 1.0. For example, if you request the availability of the content repository with

```
http://localhost:8081/actuator/metrics/coremedia.connection.contentrepositoryavailable
```

you get a response like

```
{
  "name": "coremedia.connection.contentrepositoryavailable",
  "description": "whether the content repository is currently available",
  "baseUnit": null,
  "measurements": [
    {
      "statistic": "VALUE",
      "value": 1.0
    }
  ],
  "availableTags": []
}
```

4.10.6.3 Content Feeder Metrics

The *Content Feeder* provides further metrics that are described in section "Reference | Content Feeder Metrics" of the *Search Manual*.

4.10.6.4 Workflow Server Metrics

The *Workflow Server* provides the metric `workflow.processes`, which returns the number of open workflow processes, which are processes in state `not started`, `running`, or `suspended`. The tag `definition` can be used to filter by process definition name.

For example, you can request the number of StudioTwoStepPublication workflows with

```
http://localhost:40381/actuator/metrics/workflow.processes
  ?tag=definition:StudioTwoStepPublication
```

The response:

```
{
  "name": "workflow.processes",
  "description": "Number of open process instances",
  "baseUnit": "process instances",
  "measurements": [
    {
      "statistic": "VALUE",
      "value": 1
    }
  ],
  "availableTags": []
}
```

4.10.7 Content Server Runlevel Endpoint

The runlevel endpoint provides the possibility to switch the runlevel on a content server.

Endpoint properties:

```
management.endpoint.runlevel.enabled=true
```

Endpoint URL:

```
http://localhost:8081/actuator/runlevel
```

When requested using a GET HTTP request, the endpoint will respond which runlevel is currently active.

Response:

```
{
  "RUNLEVEL": "ONLINE"
}
```

To switch the runlevel, send a POST request to the endpoint with the desired runlevel and a grace period for the switch.

Switch runlevel:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"runlevel": "MAINTENANCE", "gracePeriod": 30}' \
     http://localhost:8081/actuator/runlevel
```

4.10.8 Content Server Blob Collector Endpoint

The `blobcollector` endpoint provides the possibility to suspend the deletion of unused blobs at runtime. This is a required step for the backup of custom blob stores, as described in the section "Backup Strategy" of the Content Server Manual. Alternatively, blob deletion can be suspended with configuration property `sql.store.collector.suspend` but that requires a restart of the Content Server.

Endpoint properties:

```
management.endpoint.blobcollector.enabled=true
```

Endpoint URL:

```
http://localhost:8081/actuator/blobcollector
```

When requested using a GET HTTP request, the endpoint will respond with the current state of the blob collector. It returns `true` for the key `"suspend"`, if blob deletion is currently suspended or was requested to suspend.

Response:

```
{
  "suspend": false
}
```

To suspend blob deletion, send a POST request with `"suspend": true` to the endpoint:

Suspend blob deletion:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"suspend": true}' \
     http://localhost:8081/actuator/blobcollector
```

To resume blob deletion, send a POST request with `"suspend": false` to the endpoint:

Resume blob deletion:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"suspend": false}' \
     http://localhost:8081/actuator/blobcollector
```

4.10.9 Replicator Endpoint

The replicator endpoint can be used to enable or disable the replication process.

endpoint properties:

```
management.endpoint.replicator.enabled=true
```

Endpoint url:

```
http://localhost:8081/actuator/replicator
```

When requested using a GET HTTP request, the endpoint will respond with the state of the replicator.

Response:

```
{
  "serviceState": "Running"
}
```

Possible states are `Running`, `Stopped`, `Failed`, `Disabled` and `Unknown`.

To enable or disable the replicator, send a POST request.

Disable replicator:

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"enable": "false"}' \
     http://localhost:8081/actuator/replicator
```

4.10.10 CAE Feeder Reindex Endpoint

The reindex endpoint on the CAE Feeder can be used to reindex documents for the CAE search.

CAUTION

Please be advised, that reindexing is a very computing intensive operation and should be used with care.

Endpoint properties:

```
management.endpoint.reindex.enabled=true
```

Endpoint URL:

```
http://localhost:8081/actuator/reindex
```

For a detailed description how to use this endpoint, see the section about partial reindexing in the search manual.

4.10.11 Content Feeder Reindex Endpoint

The reindex endpoint on the Content Feeder can be used to reindex documents for the Studio search

CAUTION

Please be advised, that reindexing is a very computing intensive operation and should be used with care.

Endpoint properties:

```
management.endpoint.reindex.enabled=true
```

Endpoint URL:

```
http://localhost:8081/actuator/reindex
```

For a detailed description how to use this endpoint, see the section about partial reindexing in the search manual.

4.10.12 CAE Link Handlers Endpoint

The linkhandlers endpoint on the CAE exposes information about beans annotated with `com.coremedia.objectserver.web.links.Link` or `com.coremedia.objectserver.web.links.LinkPostProcessor`. It complements the predefined spring boot mappings endpoint.

5. Monitoring

This chapter describes how to monitor CoreMedia CMS and apply health checks and alerts. This chapter will not describe details of any specific monitoring solution, but instead provides enough details, so that you can configure your monitoring tool accordingly to intervene or to raise an alarm if required.

Note, that this chapter is focused on *CoreMedia Content Cloud*. To monitor other external systems like databases, for instance, see the corresponding documentation provided by the manufacturer.

Structure

5.1 General Concepts

In this section you will get to know about general monitoring concepts of *CoreMedia Content Cloud*.

5.1.1 Term Definitions

The following terms are used within this chapter:

Alert

Automated alerts draw human attention to a particular system if a problem has been identified which requires human interaction. Alerts are often reported via email or messaging systems.

Typical Alerts are configured based on states, thresholds or trends.

Ideally monitoring does not raise false positive alerts as important alerts may be overseen if the noise is too high. Because of this many alerts are configured with some grace period between the detection of a problem and triggering an alert.

Attribute

Typically, attributes either refer to configuration or to a service state. Examples for attributes are configured JDBC URLs, feature flags or the current runlevel of a server.

Alerts on configuration attributes typically signal a misconfiguration of the system. Alerts on state attributes are for example triggered, if a monitored service does not reach a desired state after start.

All Boolean values are attributes, as they either represent a configuration or a state.

Counter

A specific metric with a value which may increase over time. Examples for counters are an uptime in seconds or the number of received events.

Alerts on counters typically signal an imminent overflow and will typically not vanish without administrative intervention. Alerts are typically configured, so that they raise an alarm some time before the actual overflow happens.

Other possible alerts monitor a given time span and raise an alarm when either nothing happened for a long time or the counter suddenly increases drastically.

Gauge A specific metric with a value which may go up and down over time. Examples for gauges are memory usage, number of pending events or current cache size.

Alerts on gauges typically signal an overload of the system or if expected load is missing. They may vanish without administrative intervention. Typical alerts on gauges add some grace period before an alarm is raised.

Metric A metric is a measurable value which may change over time. It is either a counter which will increase over time or a gauge which may go up and down over time. Typical examples are event counters or resource consumption.

A metric typically is expressed in a given unit and a distance can be defined between two values.

Regarding Boolean values similar definitions apply. A system configuration is an attribute, while a Boolean value signaling some state which may change back and forth over time is a gauge.

5.1.2 Endpoints

Most CoreMedia systems provide Java Management Extensions (JMX) to monitor system states. Others may provide a REST API to query the state. The Spring Boot metrics actuator endpoint also exposes some metrics that can be used for monitoring, see Section 4.10.6, “Metrics Endpoint” [69].

In order to configure available managed beans (MBeans) for JMX, components need to register them via `com.coremedia.jmx.MBeanRegistrator` as described in ????.

Note, that the description of available MBeans in this chapter is based on the default *Blueprint* configuration and may vary in your deployment.

JMX Monitoring Recommendations

This manual groups monitoring on service/application level as well as by JMX MBeans. However, if your monitoring solution offers service discovery, it is recommended to use this feature rather than configuring monitoring for each service independently. So, for example instead of configuring monitoring for *CoreMedia Studio* and *Content Application Engine* for a healthy `CapConnection` you will rather search for services exposing the `CapConnection` MBean.

5.2 Monitoring Services

This section describes monitoring grouped by services of *CoreMedia Content Cloud*. For monitoring via service discovery you may instead read Section 5.3, “JMX” [86].

5.2.1 CAE Feeder

In this section you will find information about monitoring the health of the CAE Feeder.

JMX MBeans

The MBeans available for CAE Feeder are described in Section 6.4, “CAE Feeder JMX Managed Beans” in *Search Manual*.

The MBeans for CAE Feeder are by default configured via the artifact `com.coremedia.cms:caefeeder-base-component`. Contained MBeans are:

- `com.coremedia.cache.management.CacheManager`
- `com.coremedia.cap.common.CapConnectionManager`
- `com.coremedia.cap.persistentcache.dependencycache.PersistentDependencyCacheManagement`
- `com.coremedia.cap.persistentcache.proactive.HealthManager`
- `com.coremedia.cap.persistentcache.proactive.KeyManagerManagement`
- `com.coremedia.cap.persistentcache.proactive.ProactiveEngineManagement`
- `com.coremedia.cap.persistentcache.proactive.content.ContentTriggerManager`

Recommended JMX Monitoring

- Section 5.3.1, “CapConnection” [86]

  When monitoring by service, health checks for `WorkflowRepository` and `CapListRepository` are irrelevant.
- Section 5.3.5, “Proactive Engine Sub Component” [90]
- Section 5.3.4, “Health (Proactive Engine)” [90]

5.2.2 Content Application Engine

In this section you will find information about monitoring the health of *Content Application Engine*.

JMX MBeans

The MBeans for *Content Application Engine* are by default configured via artifact `com.coremedia.cms:cae-component`. Contained MBeans are:

- `com.coremedia.cache.management.CacheManager`
- `com.coremedia.cap.common.CapConnectionManager`
- `com.coremedia.objectserver.beans.ContentBeanFactory-Manager`
- `com.coremedia.objectserver.dataviews.AbstractDataView-FactoryManager`
- `com.coremedia.objectserver.view.resolver.Tem-plateViewRepositoryProviderManagement`
- `com.coremedia.objectserver.view.resolver.ViewResolver-Management`
- `com.coremedia.objectserver.web.links.LinkFormatterMan-ager`

For more details have a look at Section 5.7, "Managed Properties" in *Content Application Developer Manual*.

Recommended JMX Monitoring

- Section 5.3.1, "CapConnection" [86]

  When monitoring by service, health checks for `WorkflowRepository` and `CapListRepository` are irrelevant, because neither `WorkflowRepository` nor `CapListRepository` are required and thus the corresponding health checks will always answer `true`.

5.2.3 Content Feeder

In this section you will find information about monitoring the health of the Content Feeder.

JMX MBeans

The MBeans available for Content Feeder are described in Section 6.3, "Content Feeder JMX Managed Beans" in *Search Manual*.

Recommended JMX Monitoring

- Section 5.3.3, "Feeder" [89]

5.2.4 Content Management Server

In this section you will find information about monitoring the health of the Content Management Server.

JMX MBeans

The MBeans available for Content Management Server are described in Section 5.2, "Managed Properties" in *Content Server Manual*.

Recommended JMX Monitoring

- Section 5.3.2, "ContentServer" [87]

5.2.5 Master Live Server

In this section you will find information about monitoring the health of Master Live Server.

JMX MBeans

The MBeans available for Master Live Server are described in Section 5.2, "Managed Properties" in *Content Server Manual*.

Recommended JMX Monitoring

- Section 5.3.2, "ContentServer" [87]

5.2.6 Replication Live Server

In this section you will find information about monitoring the health of Replication Live Server.

JMX MBeans

The MBeans available for Replication Live Server are described in Section 5.2, "Managed Properties" in *Content Server Manual*.

Recommended JMX Monitoring

- Section 5.3.2, "ContentServer" [87]
- Section 5.3.6, "Replicator" [91]

5.2.7 Studio

In this section you will find information about monitoring the health of *CoreMedia Studio*, or more specifically its REST backend.

JMX MBeans

The MBeans for *CoreMedia Studio* are by default configured via artifact `com.coremedia.ui:editing-rest-component`. Contained MBeans are:

- `com.coremedia.cache.management.CacheManager`
- `com.coremedia.cap.common.CapConnectionManager`

Recommended JMX Monitoring

- Section 5.3.1, "CapConnection" [86]

Theme Importer Monitoring

The Theme Importer is either used explicitly via command line interface or implicitly through the Frontend Development Workflow. Its communication endpoint is deployed with *CoreMedia Studio*. The communication endpoint provides a simple health check with a response as JSON, to check if the endpoint is available in general. Note, that the

path below is a relative path to *CoreMedia Studio* and needs to be adjusted according to your deployment.

Find more details in Section 6.6.4, "Theme Importer" in *Frontend Developer Manual*.

5.2.8 User Changes Application

In this section you will find information about monitoring the health of the User Changes application.

JMX MBeans

The User Changes application contains the following MBeans by default:

- `com.coremedia.cache.management.CacheManager`
- `com.coremedia.cap.common.CapConnectionManager`

Recommended JMX Monitoring

- Section 5.3.1, "CapConnection" [86]

5.2.9 Workflow Server

In this section you will find information about monitoring the health of the Workflow Server.

JMX MBeans

The MBeans available for Workflow Server are described in Section 6.1.3, "Managed Properties" in *Workflow Manual*.

Recommended JMX Monitoring

- Section 5.3.1, "CapConnection" [86]

  When monitoring by service, health checks for `WorkflowRepository` are irrelevant, because there is no other `WorkflowRepository` required by this `WorkflowRepository` and thus the corresponding health checks will always answer `true`.

5.3 JMX

This section describes monitoring grouped by available Java Management Extensions (JMX) MBeans of *CoreMedia Content Cloud*. For all available MBeans and to get informed on relevant values for certain services, have a look at Section 5.2, "Monitoring Services" [81].

If supported by your monitoring solution, it is recommended to implement monitoring by service discovery. So, instead of monitoring a service explicitly, you will monitor for example all services exposing a `CapConnection` MBean instead.

5.3.1 CapConnection

The attributes and metrics mentioned in Table 5.1, "CapConnection JMX Monitoring" [86] are suggested for monitoring.

For a complete overview of available attributes and metrics have a look at `com.coremedia.cap.common.CapConnectionManager`.

`CapConnection.CapListRepositoryHealthy`	
Type	Attribute
Value Type	Boolean
Description	Signals if the caplist repository is healthy. This is a value derived from `CapListRepositoryRequired` and `CapListRepositoryAvailable` (see `JavaDoc` for details). A value of `false` signals, that the required repository is unavailable.

`CapConnection.ContentRepositoryHealthy`	
Type	Attribute
Value Type	Boolean
Description	Signals if the content repository is healthy. This is a value derived from `ContentRepositoryRequired` and `ContentRepositoryAvailable` (see `JavaDoc` for details). A value of `false` signals, that the required repository is unavailable.

`CapConnection.WorkflowRepositoryHealthy`	
Type	Attribute
Value Type	Boolean
Description	Signals if the workflow repository is healthy. This is a value derived from `WorkflowRepositoryRequired` and `WorkflowRepositoryAvailable` (see `JavaDoc` for details). A value of `false` signals, that the required repository is unavailable.

Table 5.1. CapConnection JMX Monitoring

5.3.2 ContentServer

The attributes and metrics mentioned in Table 5.2, "ContentServer JMX Monitoring" [87] are suggested for monitoring.

For a complete overview of available attributes and metrics have a look at Section 5.2, "Managed Properties" in *Content Server Manual*.

`Server.LicenseValidUntilHard`	
Type	Gauge
Value Type	long
Unit	milliseconds
Description	Time in epoch milliseconds when a license will expire and thus servers will fail to start. Note, that it is recommended to monitor `Server.LicenseValidUntilSoft` instead.

`Server.LicenseValidUntilSoft`	
Type	Gauge
Value Type	long
Unit	milliseconds

Description	Time in epoch milliseconds when a license warning will be filed to the logs. You should monitor this limit and raise and raise an alert if your license will expire soon.

`Server.RepositorySequenceNumber`

Type	Counter
Value Type	long
Description	The sequence number of the latest successful repository transaction, useful for comparison between Master Live Server and Replication Live Server. A typical health check monitors `Server.RepositorySequenceNumber` versus `Replicator.LatestIncomingSequenceNumber`, so that they do not diverge over a given threshold. A possible threshold could be the `Server.RepositorySequenceNumber` from some minutes ago, which should not be greater than `Replicator.LatestIncomingSequenceNumber`. Please consult your monitoring solution if it is possible to express such condition.

`Server.RunLevel`

Type	Attribute
Value Type	String
Description	The current run level of a server. For details on available run levels see Section 2.4, "Server Run Levels" in *Content Server Manual*. Possible values are: • `offline` • `maintenance` • `administration` • `online` A standard server should reach run level *online* after some given grace period.

`Server.RunLevelNumeric`

Type	Attribute
Value Type	int
Description	The current run level of a server. For details on available run levels see Section 2.4, "Server Run Levels" in *Content Server Manual*. Possible values are:

- 0 = `offline`
- 1 = `maintenance`
- 2 = `administration`
- 3 = `online`

A standard server should reach run level *online* after some given grace period.

Table 5.2. ContentServer JMX Monitoring

5.3.3 Feeder

The attributes and metrics mentioned in Table 5.3, "Content Feeder JMX Monitoring" [89] are suggested for monitoring.

For a complete overview of available attributes and metrics have a look at Section 6.3, "Content Feeder JMX Managed Beans" in *Search Manual*.

`Feeder.State`	
Type	Attribute
Value Type	String
Description	The state of the Content Feeder which is one of the following: • `stopped` • `starting` • `initializing` • `running` • `failed` A typical health check monitors that the Content Feeder reaches state `running` after Content Feeder startup with a certain grace period.

`Feeder.StateNumeric`	
Type	Attribute
Value Type	int
Description	The state of the Content Feeder as number which is one of the following: • 0 = `stopped`

- 1 = `starting`
- 2 = `initializing`
- 3 = `running`
- 4 = `failed`

A typical health check monitors that the Content Feeder reaches state `running` after Content Feeder startup with a certain grace period.

Table 5.3. Content Feeder JMX Monitoring

5.3.4 Health (Proactive Engine)

The attributes and metrics mentioned in Table 5.4, "CAE Feeder/Proactive Engine JMX Monitoring" [90] are suggested for monitoring.

For a complete overview of available attributes and metrics have a look at `com.coremedia.cap.persistentcache.proactive.HealthManager`.

`Health.Healthy`	
Type	Attribute
Value Type	Boolean
Description	Signals if the component is healthy in relation to the configuration. A typical health check monitors that the CAE Feeder reports that it is healthy after CAE Feeder startup with a certain grace period.

Table 5.4. CAE Feeder/Proactive Engine JMX Monitoring

5.3.5 Proactive Engine Sub Component

The Proactive Engine is a sub component of the CAE Feeder. The attributes and metrics mentioned in Table 5.5, "Proactive Engine JMX Monitoring" [91] are suggested for monitoring.

For a complete overview of available attributes and metrics have a look at Section 6.4, "CAE Feeder JMX Managed Beans" in *Search Manual* and `com.core-`

`media.cap.persistentcache.proactive.ProactiveEngineMan-agement`.

`ProactiveEngine.KeysCount`	
Type	Gauge
Value Type	int
Description	The total number of "keys" that need to be kept up-to-date by the CAE Feeder. The value may go down when destroying content or moving content outside the path configured for feeding. Should be monitored together with `ValuesCount`. See description of `ValuesCount` for details.

`ProactiveEngine.ValuesCount`	
Type	Gauge
Value Type	int
Description	The number of "keys" whose latest evaluation is still up-to-date. This is a subset of the total number of keys returned by attribute `KeysCount` and thus `ValuesCount` is always less than or equal to `KeysCount`. The value may go down on invalidations. Monitoring typically observes the difference of `KeysCount` versus `ValuesCount`: A stable state is reached, when `KeysCount` is equal to `ValuesCount`. Otherwise, so if `ValuesCount` is less than `KeysCount`, the value of `ValuesCount` should increase over time. If it does not increase within a given amount of time, you should raise an alarm.

Table 5.5. Proactive Engine JMX Monitoring

5.3.6 Replicator

The attributes and metrics mentioned in Table 5.6, “Replicator JMX Monitoring” [92] are suggested for monitoring.

For a complete overview of available attributes and metrics have a look at Section 5.2, "Managed Properties" in *Content Server Manual*.

`Replicator.LatestIncomingSequenceNumber`	
Type	Counter
Value Type	long
Description	The sequence number of the latest incoming event, useful for comparison between Master Live Server and Replication Live Server. A typical health check monitors `Server.RepositorySequenceNumber` versus `Replicator.LatestIncomingSequenceNumber`, so that they do not diverge over a given threshold. A possible threshold could be the `Server.RepositorySequenceNumber` from some minutes ago, which should not be greater than `Replicator.LatestIncomingSequenceNumber`. Please consult your monitoring solution if it is possible to express such condition.

Table 5.6. Replicator JMX Monitoring

5.4 See Also

You will find additional documentation in the following sections:

- Section 4.9, “JMX Management” [56]
- ????
- Section 5.2, “Managed Properties” in *Content Server Manual*
- Section 5.7, “Managed Properties” in *Content Application Developer Manual*
- Section 6.1.3, “Managed Properties” in *Workflow Manual*
- Section 6.4, “CAE Feeder JMX Managed Beans” in *Search Manual*
- Section 6.3, “Content Feeder JMX Managed Beans” in *Search Manual*

Glossary

Blob	Binary Large Object or short blob, a property type for binary objects, such as graphics.
CaaS	Content as a Service or short caas, a synonym for the CoreMedia Headless Server.
CAE Feeder	Content applications often require search functionality not only for single content items but for content beans. The *CAE Feeder* makes content beans searchable by sending their data to the *Search Engine*, which adds it to the index.
Content Application Engine (CAE)	The *Content Application Engine* (*CAE*) is a framework for developing content applications with *CoreMedia CMS*. While it focuses on web applications, the core frameworks remain usable in other environments such as standalone clients, portal containers or web service implementations. The CAE uses the Spring Framework for application setup and web request processing.
Content Bean	A content bean defines a business oriented access layer to the content, that is managed in *CoreMedia CMS* and third-party systems. Technically, a content bean is a Java object that encapsulates access to any content, either to CoreMedia CMS content items or to any other kind of third-party systems. Various CoreMedia components like the CAE Feeder or the data view cache are built on this layer. For these components the content beans act as a facade that hides the underlying technology.
Content Delivery Environment	The *Content Delivery Environment* is the environment in which the content is delivered to the end-user. It may contain any of the following modules: • *CoreMedia Master Live Server* • *CoreMedia Replication Live Server* • *CoreMedia Content Application Engine* • *CoreMedia Search Engine* • *Elastic Social* • *CoreMedia Adaptive Personalization*

Content Feeder	The *Content Feeder* is a separate web application that feeds content items of the CoreMedia repository into the *CoreMedia Search Engine*. Editors can use the *Search Engine* to make a full text search for these fed items.
Content item	In *CoreMedia CMS*, content is stored as self-defined content items. Content items are specified by their properties or fields. Typical content properties are, for example, title, author, image and text content.
Content Management Environment	The *Content Management Environment* is the environment for editors. The content is not visible to the end user. It may consist of the following modules: • *CoreMedia Content Management Server* • *CoreMedia Workflow Server* • *CoreMedia Importer* • *CoreMedia Site Manager* • *CoreMedia Studio* • *CoreMedia Search Engine* • *CoreMedia Adaptive Personalization* • *CoreMedia Preview CAE*
Content Management Server	Server on which the content is edited. Edited content is published to the Master Live Server.
Content Repository	*CoreMedia CMS* manages content in the Content Repository. Using the Content Server or the UAPI you can access this content. Physically, the content is stored in a relational database.
Content Server	*Content Server* is the umbrella term for all servers that directly access the CoreMedia repository: *Content Servers* are web applications running in a servlet container. • *Content Management Server* • *Master Live Server* • *Replication Live Server*
Content type	A content type describes the properties of a certain type of content. Such properties are for example title, text content, author, ...
Contributions	Contributions are tools or extensions that can be used to improve the work with *CoreMedia CMS*. They are written by CoreMedia developers - be it clients, partners or CoreMedia employees. CoreMedia contributions are hosted on Github at https://github.com/coremedia-contributions.
Control Room	*Control Room* is a *Studio* plugin, which enables users to manage projects, work with workflows, and collaborate by sharing content with other *Studio* users.
CORBA (Common Object Request Broker Architecture)	The term *CORBA* refers to a language- and platform-independent distributed object standard which enables interoperation between heterogenous applications over

	a network. It was created and is currently controlled by the Object Management Group (OMG), a standards consortium for distributed object-oriented systems. CORBA programs communicate using the standard IIOP protocol.
CoreMedia Studio	*CoreMedia Studio* is the working environment for business specialists. Its functionality covers all the stages in a web-based editing process, from content creation and management to preview, test and publication. As a modern web application, *CoreMedia Studio* is based on the latest standards like Ajax and is therefore as easy to use as a normal desktop application.
Dead Link	A link, whose target does not exist.
Derived Site	A derived site is a site, which receives localizations from its master site. A derived site might itself take the role of a master site for other derived sites.
DTD	A Document Type Definition is a formal context-free grammar for describing the structure of XML entities. The particular DTD of a given Entity can be deduced by looking at the document prolog: `<!DOCTYPE coremedia SYSTEM "http://www.coremedia.com/dtd/coremedia.dtd"` There're two ways to indicate the DTD: Either by Public or by System Identifier. The System Identifier is just that: a URL to the DTD. The Public Identifier is an SGML Legacy Concept.
Elastic Social	*CoreMedia Elastic Social* is a component of *CoreMedia CMS* that lets users engage with your website. It supports features like comments, rating, likings on your website. *Elastic Social* is integrated into *CoreMedia Studio* so editors can moderate user generated content from their common workplace. *Elastic Social* bases on NoSQL technology and offers nearly unlimited scalability.
EXML	EXML is an XML dialect used in former CoreMedia Studio version for the declarative development of complex Ext JS components. EXML is Jangaroo 2's equivalent to Apache Flex (formerly Adobe Flex) MXML and compiles down to ActionScript. Starting with release 1701 / Jangaroo 4, standard MXML syntax is used instead of EXML.
Folder	A folder is a resource in the CoreMedia system which can contain other resources. Conceptually, a folder corresponds to a directory in a file system.
Headless Server	CoreMedia Headless Server is a CoreMedia component introduced with CoreMedia Content Cloud which allows access to CoreMedia content as JSON through a GraphQL endpoint. The generic API allows customers to use CoreMedia CMS for headless use cases, for example delivery of pure content to Native Mobile Applications, Smart-

	watches/Wearable Devices, Out-of-Home or In-Store Displays or Internet-of-Things use cases.
Home Page	The main entry point for all visitors of a site. Technically it is often referred to as root document and also serves as provider of the default layout for all subpages.
IETF BCP 47	Document series of *Best current practice* (BCP) defined by the Internet Engineering Task Force (IETF). It includes the definition of IETF language tags, which are an abbreviated language code such as en for English, pt-BR for Brazilian Portuguese, or nan-Hant-TW for Min Nan Chinese as spoken in Taiwan using traditional Han characters.
Importer	Component of the CoreMedia system for importing external content of varying format.
IOR (Interoperable Object Reference)	A CORBA term, *Interoperable Object Reference* refers to the name with which a CORBA object can be referenced.
Jangaroo	*Jangaroo* is a JavaScript framework developed by CoreMedia that supports TypeScript (formerly MXML/ActionScript) as an input language which is compiled down to JavaScript compatible with Ext JS. You will find detailed descriptions on the Jangaroo webpage http://www.jangaroo.net. Jangaroo 4 is the ActionScript/MXML/Maven based version for CMCC 10. Since CMCC 11 (2110), Jangaroo uses TypeScript and is implemented as a *Node.js* and *npm* based set of tools.
Java Management Extensions (JMX)	The Java Management Extensions is an API for managing and monitoring applications and services in a Java environment. It is a standard, developed through the Java Community Process as JSR-3. Parts of the specification are already integrated with Java 5. JMX provides a tiered architecture with the instrumentation level, the agent level and the manager level. On the instrumentation level, MBeans are used as managed resources.
JSP	JSP (Java Server Pages) is a template technology based on Java for generating dynamic HTML pages. It consists of HTML code fragments in which Java code can be embedded.
Locale	Locale is a combination of country and language. Thus, it refers to translation as well as to localization. Locales used in translation processes are typically represented as IETF BCP 47 language tags.
Master Live Server	The *Master Live Server* is the heart of the *Content Delivery Environment*. It receives the published content from the *Content Management Server* and makes it available to the *CAE*. If you are using the *CoreMedia Multi-Master Management Extension* you may use multiple *Master Live Server* in a CoreMedia system.
Master Site	A master site is a site other localized sites are derived from. A localized site might itself take the role of a master site for other derived sites.
MIME	With Multipurpose Internet Mail Extensions (MIME), the format of multi-part, multimedia emails and of web documents is standardised.

MXML	MXML is an XML dialect used by Apache Flex (formerly Adobe Flex) for the declarative specification of UI components and other objects. Up to CMCC 10 (2107), CoreMedia Studio used the Open Source compiler Jangaroo 4 to translate MXML and ActionScript sources to JavaScript that is compatible with Ext JS 7. Starting with CMCC 11 (2110), a new, Node.js and npm based version of Jangaroo is used that supports standard TypeScript syntax instead of MXML/ActionScript, still compiling to Ext JS 7 JavaScript.
Personalisation	On personalised websites, individual users have the possibility of making settings and adjustments which are saved for later visits.
Projects	With projects you can group content and manage and edit it collaboratively, setting due dates and defining to-dos. Projects are created in the Control Room and managed in project tabs.
Property	In relation to CoreMedia, properties have two different meanings: In CoreMedia, content items are described with properties (content fields). There are various types of properties, e.g. strings (such as for the author), Blobs (e.g. for images) and XML for the textual content. Which properties exist for a content item depends on the content type. In connection with the configuration of CoreMedia components, the system behavior of a component is determined by properties.
Replication Live Server	The aim of the *Replication Live Server* is to distribute load on different servers and to improve the robustness of the *Content Delivery Environment*. The *Replication Live Server* is a complete Content Server installation. Its content is an replicated image of the content of a *Master Live Server*. The *Replication Live Server* updates its database due to change events from the *Master Live Server*. You can connect an arbitrary number of *Replication Live Servers* to the *Master Live Server*.
Resource	A folder or a content item in the CoreMedia system.
ResourceURI	A ResourceUri uniquely identifies a page which has been or will be created by the *Active Delivery Server*. The ResourceUri consists of five components: Resource ID, Template ID, Version number, Property names and a number of key/value pairs as additional parameters.
Responsive Design	Responsive design is an approach to design a website that provides an optimal viewing experience on different devices, such as PC, tablet, mobile phone.
Site	A site is a cohesive collection of web pages in a single locale, sometimes referred to as localized site. In *CoreMedia CMS* a site especially consists of a site folder, a site indicator and a home page for a site. A typical site also has a master site it is derived from.
Site Folder	All contents of a site are bundled in one dedicated folder. The most prominent document in a site folder is the site indicator, which describes details of a site.

Site Indicator	A site indicator is the central configuration object for a site. It is an instance of a special content type, most likely `CMSite`.
Site Manager	Swing component of CoreMedia for editing content items, managing users and workflows. The Site Manager is deprecated for editorial use.
Site Manager Group	Members of a site manager group are typically responsible for one localized site. Responsible means that they take care of the contents of that site and that they accept translation tasks for that site.
Template	In CoreMedia, JSPs used for displaying content are known as Templates. OR In *Blueprint* a template is a predeveloped content structure for pages. Defined by typically an administrative user a content editor can use this template to quickly create a complete new page including, for example, navigation, predefined layout and even predefined content.
Translation Manager Role	Editors in the translation manager role are in charge of triggering translation workflows for sites.
User Changes web application	The *User Changes* web application is a *Content Repository* listener, which collects all content, modified by *Studio* users. This content can then be managed in the *Control Room*, as a part of projects and workflows.
Variants	Most of the time used in context of content variants, variants refer to all localized versions within the complete hierarchy of master and their derived sites (including the root master itself).
Version history	A newly created content item receives the version number 1. New versions are created when the content item is checked in; these are numbered in chronological order.
Weak Links	In general *CoreMedia CMS* always guarantees link consistency. But links can be declared with the *weak* attribute, so that they are not checked during publication or withdrawal. Caution! Weak links may cause dead links in the live environment.
Workflow	A workflow is the defined series of tasks within an organization to produce a final outcome. Sophisticated applications allow you to define different workflows for different types of jobs. So, for example, in a publishing setting, a document might be automatically routed from writer to editor to proofreader to production. At each stage in the workflow, one individual or group is responsible for a specific task. Once the task is complete, the workflow software ensures that the individuals responsible for the next task are notified and receive the data they need to execute their stage of the process.

Workflow Server	The *CoreMedia Workflow Server* is part of the Content Management Environment. It comes with predefined workflows for publication and global-search-and-replace but also executes freely definable workflows.
XLIFF	XLIFF is an XML-based format, standardized by OASIS for the exchange of localizable data. An XLIFF file contains not only the text to be translated but also metadata about the text. For example, the source and target language. *CoreMedia Studio* allows you to export content items in the XLIFF format and to import the files again after translation.

Index