

COREMEDIA CONTENT CLOUD

Operations Basics



Copyright CoreMedia GmbH © 2021

CoreMedia GmbH

Ludwig-Erhard-Straße 18

20459 Hamburg

International

All rights reserved. No part of this manual or the corresponding program may be reproduced or copied in any form (print, photocopy or other process) without the written permission of CoreMedia GmbH.

Germany

Alle Rechte vorbehalten. CoreMedia und weitere im Text erwähnte CoreMedia Produkte sowie die entsprechenden Logos sind Marken oder eingetragene Marken der CoreMedia GmbH in Deutschland. Alle anderen Namen von Produkten sind Marken der jeweiligen Firmen.

Das Handbuch bzw. Teile hiervon sowie die dazugehörigen Programme dürfen in keiner Weise (Druck, Fotokopie oder sonstige Verfahren) ohne schriftliche Genehmigung der CoreMedia GmbH reproduziert oder vervielfältigt werden. Unberührt hiervon bleiben die gesetzlich erlaubten Nutzungsarten nach dem UrhG.

Licenses and Trademarks

All trademarks acknowledged.
March 23, 2021 (Release 2010)

- 1. Introduction 1
 - 1.1. Audience 2
 - 1.2. Typographic Conventions 3
 - 1.3. CoreMedia Services 5
 - 1.3.1. Registration 5
 - 1.3.2. CoreMedia Releases 6
 - 1.3.3. Documentation 6
 - 1.3.4. CoreMedia Training 10
 - 1.3.5. CoreMedia Support 10
 - 1.4. Changelog 12
- 2. Component Overview 13
 - 2.1. Architectural Overview 14
 - 2.2. Communication of Components 15
 - 2.3. Third-Party Requirements 18
- 3. System Requirements 19
 - 3.1. Java 21
 - 3.2. Databases 22
- 4. Basics of Operation 23
 - 4.1. Starting CoreMedia Command-Line Tools 24
 - 4.1.1. Configuration of the Start Routine with JPIF Files 25
 - 4.1.2. Which JVM will be used? 26
 - 4.2. CoreMedia CMS Directory Structure 27
 - 4.3. Configuration of CoreMedia Applications 28
 - 4.4. Communication between the System Applications 29
 - 4.4.1. Default Application Ports 30
 - 4.4.2. Communication Through a Firewall 30
 - 4.4.3. Binding Only a Single Network Interface 33
 - 4.4.4. Encrypting CORBA Communication Using SSL 34
 - 4.4.5. Preparing Spring Boot applications for HTTPS Connection 39
 - 4.4.6. Troubleshooting 41
 - 4.5. Collaborative Components 42
 - 4.5.1. Overview 42
 - 4.5.2. Deployment 42
 - 4.5.3. Recovery of Collaborative Components Database 43
 - 4.6. CoreMedia Licenses 45
 - 4.7. Logging 48
 - 4.7.1. Logging Configuration for Applications 48
 - 4.7.2. Logging Configuration for Apache Solr 48
 - 4.7.3. Logging Configuration for Command-Line Tools 49
 - 4.8. Security 50
 - 4.8.1. Overall Deployment 50
 - 4.8.2. Open Ports 50
 - 4.8.3. Passwords 51
 - 4.8.4. Data Storage 52
 - 4.8.5. Content Delivery 53
 - 4.8.6. Third-party Software 53
 - 4.8.7. Customizations 53
 - 4.9. CM Watchdog/Probedog 55

- 4.9.1. Starting the Watchdog/Probedog 56
- 4.9.2. Watching Databases 56
- 4.9.3. Configuration in watchdog.xml 57
- 4.9.4. Watchdog Result Codes 73
- 4.10. JMX Management 76
- 5. Monitoring 77
 - 5.1. General Concepts 78
 - 5.1.1. Term Definitions 78
 - 5.1.2. Endpoints 79
 - 5.2. Monitoring Services 80
 - 5.2.1. CAE Feeder 80
 - 5.2.2. Content Application Engine 81
 - 5.2.3. Content Feeder 81
 - 5.2.4. Content Management Server 82
 - 5.2.5. Master Live Server 82
 - 5.2.6. Replication Live Server 83
 - 5.2.7. Studio 83
 - 5.2.8. User Changes Application 84
 - 5.2.9. Workflow Server 84
 - 5.3. JMX 85
 - 5.3.1. CapConnection 85
 - 5.3.2. ContentServer 86
 - 5.3.3. Feeder 88
 - 5.3.4. Health (Proactive Engine) 89
 - 5.3.5. Proactive Engine Sub Component 89
 - 5.3.6. Replicator 90
 - 5.4. See Also 92
- Glossary 93
- Index 99

List of Figures

2.1. Architectural Overview	14
4.1. IOR inquiry and answer between CoreMedia Client and Server	29
4.2. Schema of the SSH tunnel	31

List of Tables

1.1. Typographic conventions	3
1.2. Pictographs	4
1.3. CoreMedia manuals	7
1.4. Changes	12
2.1. CoreMedia applications	15
4.1. Properties for SSH configuration	33
4.2. Properties for Single IP configuration	33
4.3. Example SSL Ports	34
4.4. Properties for Content Server SSL configuration	35
4.5. Properties for Workflow Server SSL configuration	36
4.6. Properties for Workflow to Content Server SSL configuration	37
4.7. Properties for Client ORB SSL configuration	37
4.8. Properties for persistence of collaboration data to MongoDB	42
4.9. Elements of a license file	46
4.10. Action to be used to check a certain application	55
4.11. Attributes of the Component element	57
4.12. General attributes of the Action element	58
4.13. Custom Action	59
4.14. Action DB	59
4.15. Action HTTP	61
4.16. JMX Action	62
4.17. ProcessStatus	63
4.18. Action ServerMode	64
4.19. Server Query	65
4.20. Action ServiceStatus	66
4.21. Action Script	67
4.22. WorkflowServerQuery	68
4.23. Attributes of the Edge element	69
4.24. watchdog result codes	73
5.1. CapConnection JMX Monitoring	85
5.2. ContentServer JMX Monitoring	86
5.3. Content Feeder JMX Monitoring	88
5.4. CAE Feeder/Proactive Engine JMX Monitoring	89
5.5. ProactiveEngine JMX Monitoring	90
5.6. Replicator JMX Monitoring	91

List of Examples

4.1. Output of <code>cm</code> in the <code>cms-tools</code> directory	24
4.2. A sample license file	46

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 \[13\]](#)
- Some basics about the operating environments used by *CoreMedia Content Cloud* are described in [Chapter 3, System Requirements \[19\]](#).
- the administration essentials of *CoreMedia CMS* for example how to start the applications, in [Chapter 4, Basics of Operation \[23\]](#)

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

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" \[6\]](#) 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-10>

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 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 [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 and Javadoc as PDF files and as online documentation at the following URL:

<https://documentation.coremedia.com/cmcc-10>

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 <i>Adaptive Personalization</i> , the CoreMedia module for personalized websites. You will learn how to configure the GUI used in <i>CoreMedia Studio</i> , 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	<p>This manual gives an overview over the structure and features of <i>CoreMedia Content Cloud</i>. It describes the content type model, the <i>Studio</i> extensions, folder and user rights concept and many more details. It also describes administrative tasks for the features.</p> <p>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.</p>
Connector Manuals	Developers, administrators	This manual 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 <i>Content Application Engine [CAE]</i> . 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.
Content Server Manual	Developers, architects, administrators	This manual describes the concepts and administration of the main CoreMedia component, the <i>Content Server</i> . 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.

Manual	Audience	Content
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 <i>Elastic Social</i> 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 <i>Headless Server</i> . 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 <i>Importer</i> application and how you define the transformations that convert your content into CoreMedia 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 or the usage of the watchdog component.
Search Manual	Developers, architects, administrators	This manual describes the configuration and customization of the <i>CoreMedia Search Engine</i> and the two feeder applications: the <i>Content Feeder</i> and the <i>CAE Feeder</i> .
Site Manager Developer Manual	Developers, architects, administrators	This manual describes the configuration and customization of <i>Site Manager</i> , the Java based stand-alone application for administrative tasks. You will learn how to configure the <i>Site Manager</i> with property files and

Manual	Audience	Content
		<p>XML files and how to develop your own extensions using the <i>Site Manager API</i>.</p> <p>The Site Manager is deprecated for editorial work.</p>
Studio Developer Manual	Developers, architects	This manual describes the concepts and extension of <i>CoreMedia Studio</i> . You will learn about the underlying concepts, how to use the development environment and how to customize <i>Studio</i> to your needs.
Studio User Manual	Editors	This manual describes the usage of <i>CoreMedia Studio</i> for editorial and administrative work. It also describes the usage of the <i>Adaptive Personalization</i> and <i>Elastic Social</i> GUI that are integrated into <i>Studio</i> .
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 <i>CoreMedia Unified API</i> , which is the recommended API for most applications. This includes access to the content repository, the workflow repository and the user repository.
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 <i>Workflow Server</i> . 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 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

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:

Support request

- 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.

To put it in a nutshell, CoreMedia needs:

Support checklist

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

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. The location where component log output is written is specified in its `logback.xml` file.

Log files

Which Log File?

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

Where do I Find the Log Files?

By default, log files can be found in the CoreMedia component's installation directory in `/var/logs` or for web applications in the `logs/` directory of the servlet container. See [Section 4.7, "Logging" \[48\]](#) for details.

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

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.

Overview and deployment

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.

The communication between all applications can be secured. See [Section 4.4, “Communication between the System Applications” \[29\]](#) for details.

Security

All applications of *CoreMedia CMS* use [Logback](#) for logging. See [Section 4.7, “Logging” \[48\]](#) 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 “JMXDump” in *Content Server Manual*. *CoreMedia CMS* comes with two applications specific for monitoring, that is the Probedog and the Watchdog. You can use these applications to monitor the status of CoreMedia applications and to restart a server.

Logging and Monitoring

2.1 Architectural Overview

Figure 2.1, "Architectural Overview" [14] shows a deployment of CoreMedia CMS.

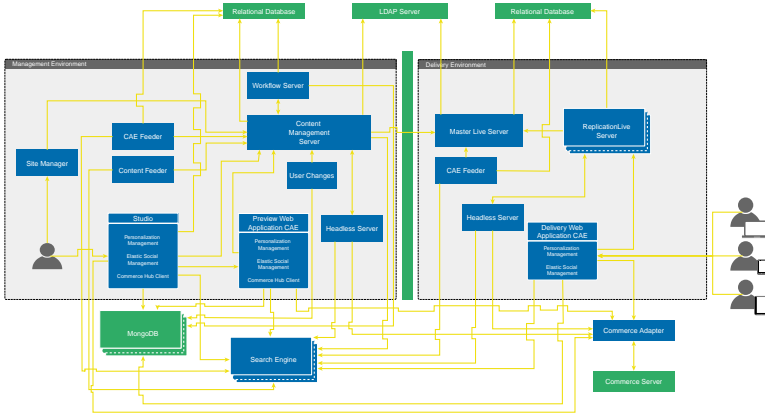


Figure 2.1. Architectural Overview

If you want to have an overview of the default ports used in CoreMedia, check out [Section 4.4.1, "Default Application Ports" \[30\]](#).

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” \[29\]](#) for details. [Table 2.1, “CoreMedia applications” \[15\]](#) 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 <i>Content Server Manual</i>)	Manages the content in the Management Environment and publishes content to the <i>Master Live Server</i> .	No	<ul style="list-style-type: none"> • 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 <i>Content Server Manual</i>)	Manages the content in the Delivery Environment	Multiple instances when Multi-Master is used	<ul style="list-style-type: none"> • All clients. • External relational database
Replication Live Server (For more information, check out: Section 2.2, “Replication Live Servers” in <i>Content Server Manual</i>)	Serves content to the CAEs	Multiple instances can be attached to one <i>Master Live Server</i>	<ul style="list-style-type: none"> • Content Application Engine • External relational database
Workflow Server (For more information, check out: Chapter 2, Overview of CoreMedia Workflow in <i>Workflow Manual</i>)	Executes workflows	No	Content Management Server
Site Manager (For more information, check out: Site Manager Developer Manual)	Management tool for workflows and users.	Yes	<ul style="list-style-type: none"> • Content Management Server • Workflow Server

Application	Purpose	Multiple Instances	Communicates with
Studio (For more information, check out: Section 2.1, "Architecture" in <i>Studio Developer Manual</i>)	Content editing and management. Hosts management extensions for <i>Elastic Social</i> and <i>Adaptive Personalization</i> .	One web application	<ul style="list-style-type: none"> • Content Management Server • Search Engine • Workflow Server • MongoDB • External relational database
Search Engine (For more information, check out: Chapter 3, <i>Search Engine</i> in <i>Search Manual</i>)	Indexes content and provides searches functionality.	Yes.	<ul style="list-style-type: none"> • Content Management Server • Content Feeder • CAE Feeder • Studio • Content Application Engine
CAE Feeder (For more information, check out: Chapter 5, <i>Searching for CAE Content Beans</i> in <i>Search Manual</i>)	Feeds content beans into the <i>Search Engine</i>	Multiple instances possible, for example when reindexing.	<ul style="list-style-type: none"> • Content Management Server • Search Engine • External relational database
Content Feeder (For more information, check out: Chapter 4, <i>Searching for Content</i> in <i>Search Manual</i>)	Serves content to the <i>Search Engine</i>	Multiple instances possible, for example when reindexing.	<ul style="list-style-type: none"> • Content Management Server • Search Engine
Content Application Engine (For more information, check out: Chapter 2, <i>Overview in Content Application Developer Manual</i>)	Serves sites to the customer. Hosts <i>Elastic Social</i> and <i>Adaptive Personalization</i> extension.	Multiple instances can be attached to one <i>Master Live Server</i> or <i>Replication Live Server</i>	<ul style="list-style-type: none"> • Content Server • MongoDB database for <i>Elastic Social</i> • Search Engine • Custom external systems
Importer (For more information, check out: Chapter 3, <i>Administra-</i>	Imports content into the <i>Content Management Server</i> .	Yes	<ul style="list-style-type: none"> • Content Management Server

Application	Purpose	Multiple Instances	Communicates with
<i>tion And Operation in Importer Manual]</i>			
Watchdog (For more information, check out: Section 4.9, "CM Watchdog/Probedog" [55])	Monitors CoreMedia applications and databases	Yes	Monitored applications

Table 2.1. CoreMedia applications

2.3 Third-Party Requirements

As shown in [Figure 2.1, “Architectural Overview” \[14\]](#) *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:

Third-party software

- 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-10>.

Installation

3. System Requirements

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:

Use only recommended systems

- *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!



You will find the approved components in the Supported Operation Environments document on <https://documentation.coremedia.com/cmcc-10>. In the following sections you will find some general hints for the usage of these components:

Supported operation environments

- Java platforms in [Section 3.1, "Java" \[21\]](#),
- Databases in [Section 3.2, "Databases" \[22\]](#),

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-10-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 probedog 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 ".jpiif", 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.jpiif`). 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" \[29\]](#).

Furthermore, the target of the log outputs of the Java process (see [Section 4.7, "Logging" \[48\]](#)) 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.jpiif` 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.jpiif` for general environment settings for the Java programs in the CoreMedia system.
- `post-config.jpiif` 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" \[24\]](#)) 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" \[48\]](#)).
- `./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

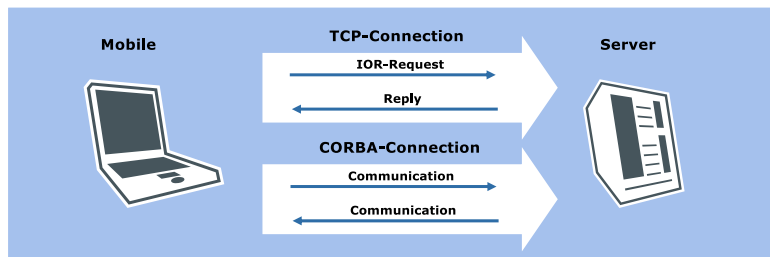


Figure 4.1. IOR inquiry and answer between CoreMedia Client and Server

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 Section 2.8.3, "Port Conventions" in *Deployment Manual*, 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" \[31\]](#) shows the scenario:

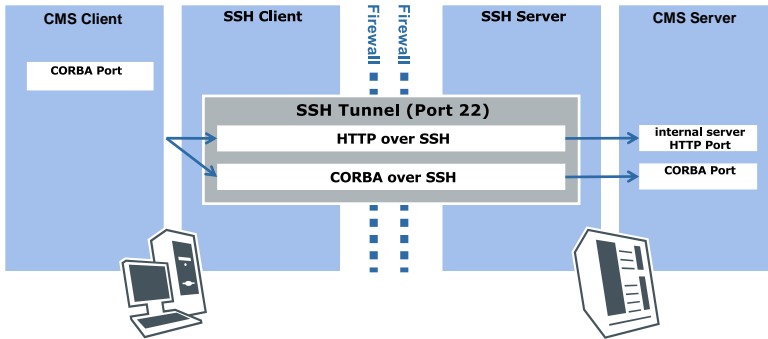


Figure 4.2. Schema of the SSH tunnel

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	<code>com.coremedia.corba.client.redirect.original-host</code>	<CMSServer>
System	<code>com.coremedia.corba.client.redirect.redirect-host</code>	<SSHClient>
System	<code>com.coremedia.corba.client.redirect.original-port</code>	<CorbaPort>
System	<code>com.coremedia.corba.client.redirect.redirect-port</code>	<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	<code>com.coremedia.corba.server.host</code>	<IpAddress>

Property Type	Property Name	Property Value
System	<code>com.coremedia.corba.server.single-ip</code>	<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" \[34\]](#) 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
<code>com.coremedia.corba.server.port</code>	14300

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

Table 4.4. Properties for Content Server SSL configuration

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

Prepare the Workflow Server for SSL communication

- Add the following properties to the workflow server's configuration:

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

Table 4.5. Properties for Workflow Server SSL configuration

- 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.

- 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
```

- 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.key store	<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.

- 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
```

- Configure the ORB for SSL by setting the properties from [Table 4.7, "Properties for Client ORB SSL configuration" \[37\]](#).

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
<code>com.coremedia.corba.client.ssl.key store</code>	<code><path to editor.keystore></code>
<code>com.coremedia.corba.client.ssl.passphrase</code>	<code><mypassword></code>

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.key store` 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.

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.

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

Property	Example	Description
		plication.properties files of <i>Studio</i> , <i>User Changes</i> and <i>Workflow Server</i> 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.

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](#).

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 “Sessions” in *Content Server Manual*) to get this information and the utility `usedlicenses` (see Section “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: <ul style="list-style-type: none"> • <i>production</i>: The Content Management Server • <i>publication</i>: The Master Live Server • <i>live</i>: The Replication Live Server

Element	Attribute	Description
Property	name	The aim of the property. Possible values are: <ul style="list-style-type: none"> 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 modify the `src/main/resources/logback-spring.xml` 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 https://lucene.apache.org/solr/guide/8_6/configuring-logging.html 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.

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"](#) [29]. 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” \[39\]](#) 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” \[29\]](#).

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” \[33\]](#) 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 factor 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 “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 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 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.

4.8.5 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.

4.8.6 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 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 CM Watchdog/Probedog

The following actions can be executed with the watchdog:

- Applications of the CoreMedia system can be monitored for functioning.
- Applications can be locally stopped and/or restarted.
- Depending on the state of the system, further actions can be triggered.

The watchdog is delivered in two flavors:

- *watchdog*
The watchdog is an independent monitoring process which is used to test regularly whether applications are functioning and, in case of error, to restart them. The watchdog will be installed as a web application.
- *probedog*
The probedog is a process for one-time checking of the status of an application. This variant is used with integration of the CoreMedia system in a high-availability cluster. The probedog can be used as a diagnostic tool for the momentary status during operation. The probedog delivers a return code which can be evaluated by a shell script. It is deployed as part of the standalone server applications respectively the server tools.

The *watchdog*'s main configuration file is the `watchdog.xml`, where the applications and actions are defined. To configure the path of this file and which of the defined applications should be watched, the following two properties can be used:

```
# the file where the applications are defined
watchdog.config=properties/corem/watchdog.xml

# a space separated list of applications that should be watched by this
watchdog
watchdog.components=WatchContentServer
```

- `watchdog.config`: Configure the location of `watchdog.xml` (default is `properties/corem/watchdog.xml`)
- `watchdog.components`: A space separated list of applications that should be watched.

The following table shows the actions you can use to monitor a specific application. In addition, you can use the `<Script>` and `<Custom>` actions to define your own monitoring. See [Section "Action Elements" \[58\]](#) for a detailed description of the actions.

Application	Action
Content Server	<ul style="list-style-type: none"> • <code><ServerQuery></code>: Checks the overall status • <code><ServiceStatus></code>: Checks the status of a specific service

Application	Action
	<ul style="list-style-type: none"> • <code><ServerMode></code>: Checks the runlevel
Workflow Server	<ul style="list-style-type: none"> • <code><WorkflowServerQuery></code>: Checks the overall status of a <i>Workflow Server</i> • <code><ProcessStatus></code>: Checks if a specified number of processes is running
Database	<ul style="list-style-type: none"> • <code><DB></code>: Checks the status of the database
CAE	<ul style="list-style-type: none"> • <code><Http></code>: Checks if the response code is "200" and if the response matches a given regular expression

Table 4.10. Action to be used to check a certain application

4.9.1 Starting the Watchdog/Probedog

Watchdog

The watchdog is always started with the servlet container.

Probedog

The Probedog can be started with: `cm probedog <component>`

4.9.2 Watching Databases

If you want to check a database directly over JDBC you need to know the connection data of the database.

For a stand-alone application, you have to install the watchdog together with the *Content Server*, so that the watchdog can use the `sql.properties` file of the *Content Server*.

A web application deployment of the watchdog has its own `sql.properties` file. By default, the settings match the database of the *Content Management Server*.

4.9.3 Configuration in watchdog.xml

An administrator configures the watchdog in the `watchdog.xml` file. The watchdog executes actions to check the watched applications and executes further actions depending on the check results. The watched applications, the actions and the mapping from actions to actions are described by XML elements. The mapping from actions to actions is defined by `<Edge/>` tags.

The general XML file structure is as follows:

```
<Watchdog>
  <Component>
    <AnActionElement/>
    <Edge/>
  </Component>
  <Component>
    <AnActionElement/>
    <Edge/>
  </Component>
</Watchdog>
```

`<AnActionElement>` is a placeholder for one of several possible action elements. `<Component>`, `<AnActionElement/>` and `<Edge/>` can occur multiple times.

In the configuration file, applications are linked with actions. In [Section "Example Configuration of a Watchdog" \[70\]](#) you will find a description of a watchdog file.

Component Element

For each watched application you need a `<Component>` element in the `watchdog.xml` file. The element has the following attributes:

Attribute	Optional	Description
<code>name</code>		A unique name.
<code>startAction</code>		The name of the action which should be executed first when the watchdog is started. If this is a simple status test (probe), it's the only action which is executed.

Attribute	Optional	Description
delay	x	The attribute specifies the delay in seconds until the action defined in the <code>startAction</code> attribute is executed.

Table 4.11. Attributes of the Component element

Action Elements

An action definition is defined by the tag `<ActionType>`. For `<ActionType>`, the name of the desired action is entered. A list of all possible actions is given in the table below. The action is configured with a series of attributes, as follows:

NOTE

Actions that need to log into the *Content Server*, such as the `ServerMode` or `ProcessStatus` action, can use the predefined user "watchdog" with the default password "watchdog" for this purpose.



General attributes

Attribute	Optional	Description
name		The name of the action. It must be unique within the configuration file.
interval	x	Interval (in seconds) at which the number of action calls is checked. Checking is switched off with "0" (Default).
events	x	Maximum permissible number of calls of the action within "interval". If the maximum number is exceeded, the action is not executed and the error message "respawning_too_fast" is delivered. "events" is only used if "interval" <> "0".
timeout	x	Time (in seconds) to wait until a pending action is canceled with the error message "11". The default value is 60 seconds.

Table 4.12. General attributes of the Action element

In addition, specific attributes can be assigned to the actions. These can be found in the following description of the actions.

Possible actions

Custom

Description This action invokes a custom watchdog action (see the Javadoc of `CustomAction` for details). You have to add two nested tags - `Custom` and `Action` - into the `watchdog.xml` file to call the action.

Example:

```
<Custom name="my first action" timeout="10">
  <Action class="com.customer.MyWatchDogAction" myprop="ok">
    <MySub class="com.customer.MySub" foo="bar" />
    <MySub class="com.customer.MySub" foo="baz" />
  </Action>
</Custom>
```

In this example, class `com.customer.MyWatchDogAction` would need to implement `CustomAction`, would need a no-args constructor, a setter for `myprop` and a method `addMySub` accepting a `MySub` object. The Class `com.customer.MySub` would need a no-args constructor and a setter for `foo`. This allows for elaborate configuration. Please read the chapter "The BeanParser" of the *Workflow Manual* for details.

Make sure that your custom action will not accumulate resources (Database, JMS connections etc.), otherwise the watchdog would itself need to be watched. In all nontrivial cases, starting an external script is strongly preferred.

The nested tags of the `Action` tag depend on the setters defined in your custom action class, here `com.customer.MyWatchdogAction`.

Attributes The `Custom` tag supports the general attributes for actions. All sub tags of `Custom` need the class attribute. All other attributes of the `Action` tag and its sub tags depend on the custom action.

Error Codes The error codes returned from the `execute()` method of the custom action class.

Table 4.13. Custom Action

DB

Description This action checks a CoreMedia database over the JDBC API.

Attribute *driver*
JDBC driver to use

DB

url

URL where to connect to the database

user

Name of the database user

password

Password of the database user

propertyFile

The CoreMedia database configuration file (`sql.properties`) where the settings (`driver`, `URL`, `password`, `user`) are configured. The path must be relative to `$COREM_HOME/properties`. See [????](#) for details of the syntax of the configuration file.

You have to specify either `driver`, `URL`, `user` and `password` or the *propertyFile* attribute.

sql

The database statement that is used to check the database. Default is `SELECT * FROM Resources WHERE id_ = 1.`

Error codes

0 (ok)

10 (unexpected_error)

11 (timeout)

12 (error)

13 (respawning_too_fast)

21 (io_error)

61 (no_jdbc_driver)

62 (no_connection)

DB

63 (sql_exception)

Table 4.14. Action DB

Http

Description This action requests a URL and checks whether the response code is "200". If so configured, it also checks whether the response matches a given regular expression.

Attribute

url
Requested URL

user
User name to use for HTTP basic authentication, if given together with the password attribute

password
Password to use for HTTP basic authentication, if given together with the user attribute

pattern
Regular expression according to `java.util.regex.Pattern` to be used for verifying the response

encoding
Character encoding for parsing the response, if the pattern attribute is given

maxSize
Maximum permitted size of the response in characters (default 65536), if the pattern attribute is given

Error codes

0 (ok)

10 (unexpected_error)

11 (timeout)

12 (error)

13 (respawning_too_fast)

Http

21 (io_error)

Table 4.15. Action HTTP

JMX

Description The JMX action retrieves an attribute from any application via JMX and converts the attribute's value into a watchdog code. This attribute conversion is done by one or more configured `com.coremedia.watchdog.impl.CodeConverter` instances. Each configured converter might either return a watchdog code or a special "not responsible" code. The JMX action iterates over each converter until a watchdog code is returned. In case that all converter return a "not responsible" code, a configured default code is used.

By default, a converter `com.coremedia.watchdog.impl.NumberRangeConverter` is provided which expects the JMX attribute to be a number (such as int or long) and which returns a configured watchdog code if the number is included in a configured range of numbers.

You might also implement a custom converter by extending `com.coremedia.watchdog.impl.CodeConverter`. See the Javadoc for more details.

Attribute

serviceUrl

A URL to connect to a JMX server

objectName

The qualified name of the MBean.

attributeName

The name of the MBean's attribute

defaultCode

The watchdog code to use if none of the converters is responsible

username

The user name for the JMX connection (optional).

password

The password for the JMX connection (optional).

ProcessStatus

	Domain of the user
	<i>password</i>
	CoreMedia user password to log on to the server
	<i>processName</i>
	The name of the process definition. The predefined workflows have the following process names: GlobalSearchAndReplace, SimplePublication, ThreeStepPublication, TwoStepApproval, TwoStepPublication.
	<i>minCount</i>
	The minimum number of instances to expect, typically 1
	<i>maxCount</i>
	The maximum number of instances to expect
Error Codes	0 (ok)
	10 (unexpected_error)
	11 (timeout)
	12 (error)
	13 (respawning_too_fast)
	21 (io_error)
	32 (invalid_login)
	91 (escalated)
	92 (too_few_instances)
	93 (too_many_instances)

Table 4.17. ProcessStatus

ServerMode

Description	This action checks the runlevel of the CoreMedia server. This ensures that the server has reached the specified runlevel and that certain clients can connect to the server.
Attribute	<i>url</i>

ServerMode

	<p>URL where to get the IOR of the server</p> <p><i>user</i></p> <p>CoreMedia user name to log on to the server.</p> <p><i>domain</i></p> <p>Domain of the user.</p> <p><i>password</i></p> <p>CoreMedia user password to log on to the server</p> <p><i>mode</i></p> <p>The expected server runlevel. Valid values are "maintenance", "administration" and "online".</p>
Error codes	<p>0 (ok)</p> <p>10 (unexpected_error)</p> <p>11 (timeout)</p> <p>12 (error)</p> <p>13 (respawning_too_fast)</p> <p>21 (io_error)</p> <p>31 (no_licenses)</p> <p>32 (invalid_login)</p> <p>33 (corba_error)</p> <p>71 (insufficient_mode)</p>

Table 4.18. Action ServerMode

ServerQuery

Description	<p>This action executes a database query on the CoreMedia server over the CORBA API. In this way, all integral components of the server from ORB to the database connection are checked.</p>
Attribute	<p><i>url</i></p>

ServerQuery

	URL where to get the IOR of the server
	<i>user</i>
	CoreMedia user name to log on to the server
	<i>domain</i>
	Domain of the user.
	<i>password</i>
	CoreMedia user password to log on to the server
Error Codes	0 [ok]
	11 [timeout]
	12 [error]
	13 [respanwing_too_fast]
	21 [io_error]
	31 [no_licenses]
	32 [invalid_login]
	33 [corba_error]
	41 [repository_error]
	51 [query_malformed]
	52 [query_failed]

Table 4.19. Server Query

ServiceStatus

Description	This action checks a certain service offered by the CoreMedia server, for example the replicator of a <i>Replication Live Server</i> .
Attribute	<i>url</i>
	URL where to get the IOR of the server
	<i>user</i>
	CoreMedia user name to log on to the server

ServiceStatus

domain

Domain of the user.

password

CoreMedia user password to log on to the server

service

Name of the checked service. Possible values are:

- replicator
- adminlogin
- userlogin

Error codes	0 (ok)
	10 (unexpected_error)
	11 (timeout)
	12 (error)
	13 (respawning_too_fast)
	21 (io_error)
	31 (no_licenses)
	32 (invalid_login)
	33 (corba_error)
	81 (service_stopped)
	82 (service_failed)
	83 (service_disabled)

Table 4.20. Action ServiceStatus

Script

Description	This action executes a shell command and checks whether the programs' return code is "0". If the command returns with a different value, an error is assumed.
-------------	---

Script

Under the Windows operating system you cannot use non-executable commands like `dir`, `copy` or `echo` directly. They will result in a `java.io.IOException`. The reason is that these commands are part of the Windows command interpreter and not a separate executable. To run these commands use the Windows command interpreter by calling `cmd.exe`, for example: `cmd.exe /C echo test`.

Attribute	<i>command</i>
	Name of the executable command

Error codes	0 (ok)
-------------	--------

Table 4.21. Action Script

WorkflowServerQuery

Description	This action executes a database query on the CoreMedia workflow server over the CORBA API. In this way, all integral components of the server from ORB to the database connection are checked.
-------------	--

Attribute	<i>url</i>	URL where to get the IOR of the server
	<i>user</i>	CoreMedia user name to log on to the server
	<i>domain</i>	Domain of the user.
	<i>password</i>	CoreMedia user password to log on to the server

Error Codes	0 (ok)
	10 (unexpected_error)
	11 (timeout)
	12 (error)
	13 (respanwing_too_fast)

WorkflowServerQuery

21 (io_error)
32 (invalid_login)
41 (repository_error)
52 (query_failed)

Table 4.22. WorkflowServerQuery

Edge Element

An edge connects two actions within a component, depending on the result code of the first action. The edges and actions form a graph. The watchdog process walks through the actions and edges in the graph. Edges are defined with `<Edge>` elements in `<Component>` elements after action elements.

Edges are configured with the following attributes:

Attribute	Optional	Description
<code>from</code>		The name of the action from which the edge starts.
<code>to</code>		The name of the action which follows.
<code>code</code>		Result code (numerical entry or symbolic name). If this value equals the result code of the action named in the <code>from</code> attribute, then the action named in the <code>to</code> attribute is executed. Edges may not start from the same action with the same error code to different target actions. To avoid defining an <code>Edge</code> tag for all error codes, the edge with the error code "Error" or "12" serves as the default edge. If an error occurs for which no edge is defined, the watchdog process will follow the edge with result code "Error".
<code>delay</code>	x	This attribute specifies the delay in seconds until the subsequent action is executed. The default value is 10 seconds.

Table 4.23. Attributes of the Edge element

Example:

If the watchdog process has to execute an action "B" in case an action "A" returns with result code "0", you have to configure:

```
<Edge from="A" to="B" code="0" / >
```

CAUTION

Component definitions should not contain cycles in which every edge has a delay of 0. Such cycles can cause high CPU loads and excessive usage of system processes and open files.



Example Configuration of a Watchdog

This section explains a simple `watchdog.xml` file configuration. The file begins with the XML and document type declarations:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<!DOCTYPE Watchdog SYSTEM "../lib/xml/coremedia-watchdog.dtd">
```

That is, the DTD for the `watchdog.xml` file with the name `coremedia-watchdog.dtd` is located on the local host under the installation directory in the `lib/xml` directory.

```
<Component name="WatchServer" startAction="WS-CorbaQuery">
```

A component called WatchServer is configured which immediately starts the action "WS-CorbaQuery".

```
<ServerQuery name="WS-CorbaQuery" url="http://localhost:8080/ior" user="watchdog" password="watchdog"/>
```

The start action WS-CorbaQuery is defined as a ServerQuery action that requests the CoreMedia server IOR URL at <http://localhost:8080/ior> and logs in with user name "watchdog" and password "watchdog".

```
<ServerQuery name="WS-CorbaReQuery" url="http://localhost:8080/ior" user="watchdog" password="watchdog"/>
```

Action `WS-CorbaReQuery` is the same type of action with the same attributes. The meaning of this double definition will become clear below in the description of the edges.

```
<DB name="WS-CheckDB" propertyFile="corem/sql.properties"/>
```

Another action to check the database. The file `corem/sql.properties` contains the connection parameters for the database.

```
<Script name="WS-RestartServer" command="service <servicename> restart " timeout="15" interval="600" events="3"/>
```

This script action restarts the CoreMedia Content Server Tomcat on a Linux system. The concrete command depends on your concrete installation. If the server has not restarted successfully after 15 seconds (`timeout` attribute), the result code is 11 (Timeout). If the server is restarted four times within 600 seconds (`interval` attribute), result code 13 (RespawningTooFast) is returned. This script action restarts the CoreMedia Content Server Tomcat on a Linux system. The concrete command depends on your concrete installation. If the server has not restarted successfully after 15 seconds (`timeout` attribute), the result code is 11 (Timeout). If the server is restarted four times within 600 seconds (`interval` attribute), result code 13 (RespawningTooFast) is returned.

```
<Script name="WS-Abort" command="echo watchdog; watch server: abort" timeout="10"/>
```

This action prints out an abort message (`command` attribute). The timeout interval for this message is 10 seconds (`timeout` attribute). Alternatively you can email to the watchdog administrator to inform him about the watchdog termination. Now connect the previously defined actions with `<Edge>` elements.

```
<Edge from="WS-CorbaQuery" to="WS-CorbaQuery" code="ok" delay="60"/>
```

In the error-free case, when the result code is `ok`, the server is queried every 60 seconds.

```
<Edge from="WS-CorbaQuery" to="WS-CorbaQuery" code="no_licenses" delay="60"/>
```

The same happens, if the result code is `no_licenses`, because there was no free license to log on to the server.

```
<Edge from="WS-CorbaQuery" to="WS-Abort" code="invalid_login" delay="0"/>
```

If the result code is "invalid_login", because the authentication has failed, then the abort action is executed. The administrator must correct the login configuration and restart the watchdog application later.

```
<Edge from="WS-CorbaQuery" to="WS-CheckDB" code="error"
delay="0"/>
```

If WS-CorbaQuery returns an error, the action WS-CheckDB is invoked immediately without delay. The latter action checks whether there is a database error. As the result code "error" is the default code, this action is also invoked for all the result codes for which no <Edge> element is configured.

```
<Edge from="WS-CheckDB" to="WS-CorbaReQuery" code="ok"
delay="0"/>
```

If the database check results in no error, the action WS-CorbaReQuery is called to check the server again. In this way, a possibly unnecessary restart of the server can be avoided. Remember that WS-CheckDB was called as a reaction to an error from WS-CorbaQuery. If the reason for this error was a database problem, the server will continue to operate without restart as soon as the database is online again. The server is restarted only if the database is OK and a following check on the server fails again.

```
<Edge from="WS-CheckDB" to="WS-CheckDB" code="error"
delay="60"/>
```

The database is checked every 60 seconds as long as the database returns an error result. As the result code "error" is the default code, this action is also invoked for all the result codes for which no <Edge> element is configured.

```
<Edge from="WS-CheckDB" to="WS-Abort" code="no_jdbc_driver"
delay="0"/>
```

If the database check fails due to a missing JDBC driver, the abort action is invoked without delay. The administrator must correct the driver configuration and restart the watchdog application later.

```
<Edge from="WS-CorbaReQuery" to="WS-CorbaQuery"
code="ok" delay="60"/>
```

If the server check results in no error, the error-free state is reached again with the action WS-CorbaQuery being called with 60 seconds delay.

```
<Edge from="WS-CorbaReQuery" to="WS-CorbaQuery"
code="no_licenses" delay="60"/>
```

The error-free state is also reached with 60 seconds delay when there are no free licenses to log on to the server.

```
<Edge from="WS-CorbaReQuery" to="WS-Abort" code="invalid_login" delay="0"/>
```

If the result code is "invalid_login", because the authentication has failed, the abort action is invoked without delay. The administrator must correct the login configuration and restart the watchdog application later.

```
<Edge from="WS-CorbaReQuery" to="WS-RestartServer" code="error" delay="0"/>
```

If the second server check yields an error again, the action "WS-RestartServer" is invoked to restart the server. At this point the database works correct and there seems to be an internal server error, which hopefully can be solved with a server restart. As the result code "error" is the default code, this action is also invoked for all the result codes for which no <Edge> element is configured.

```
<Edge from="WS-RestartServer" to="WS-CorbaQuery" code="ok" delay="60"/>
```

If the server was restarted successfully, the error-free state is reached again with the action WS-CorbaQuery being called with 60 seconds delay.

```
<Edge from="WS-RestartServer" to="WS-Abort" code="error" delay="0"/>
```

If the server restart has failed, the abort action is invoked without delay. The administrator must analyze the reason, why the server fails to start and restart the watchdog application. As the result code "error" is the default code, this action is also invoked for all the result codes for which no <Edge> element is configured.

```
<Edge from="WS-RestartServer" to="WS-Abort" code="restarting_too_fast" delay="0"/>
```

If the server is restarted more than three times in 600 seconds, the abort action is invoked without delay. The administrator must analyze the reason, why the server fails to start and restart the watchdog application later.

```
</Component>
```

4.9.4 Watchdog Result Codes

The watchdog actions return result codes. The following table lists all possible codes:

Result code	Symbolic name	Description
0	"ok"	The application works properly without errors.

Result code	Symbolic name	Description
10	"unexpected_error"	An exception has been thrown, that is not caught and handled by the action.
11	"timeout"	An action has not finished in the specified time interval.
12	"error"	An action has failed with an unknown error. This is the default code for all unhandled codes in <Edge> elements.
13	"respawning_too_fast"	An action was executed too often in a specified time interval (see Section "Action Elements" [58]).
21	"io_error"	An action has failed with an I/O error.
31	"no_licenses"	An action has failed because there was no free license.
32	"invalid_login"	An action has failed to authenticate against the CoreMedia server.
33	"corba_error"	An action has failed because a CORBA ORB raises a CORBA exception.
41	"repository_error"	An action has failed with an exception thrown in the CoreMedia server repository.
51	"query_malformed"	An action has failed because the query was malformed. This result code indicates a serious internal error and should not happen.
52	"query_failed"	An action has failed because of a database error.
61	"no_jdbc_driver"	An action has failed to load the JDBC driver class. The specified driver class is either invalid, or the JAR file is missing in the class path.
62	"no_connection"	An action has failed to establish a database connection. This indicates an internal database error or a configuration problem.

Result code	Symbolic name	Description
63	"sql_exception"	An action has failed to create or execute a SQL statement. The error indicates an internal database error or a configuration problem.
71	"insufficient_mode"	This result code indicates that the current run level of the server is lower than the configured run level
81	"service_stopped"	This result code is returned by the ServiceStatus action if the service is not running, either because it has been stopped, or because the run level of the server is too low for the service to run.
82	"service_failed"	The ServiceStatus action has failed because the service was ended due to a "critical error".
83	"service_disabled"	The ServiceStatus action has failed because the checked service is not activated. The service activation can depend on the server type. For example, the replicator service is only available on a <i>Replication Live Server</i> .
91	"code_escalated"	The ProcessStatus action has found escalated tasks in the observed process.
92	"code_toofewinstances"	The ProcessStatus action has found too few instances of the observed process definition.
93	"code_toomanyinstances"	The ProcessStatus action has found too many instances of the observed process definition.

Table 4.24. watchdog result codes

4.10 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. If you use a JDK with version 6 or higher, 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.

All *CoreMedia* web applications use a common component for JMX management, that provides common JMX infrastructures such as a remote connector server.

The remote connector server can be configured by providing the property `management.server.remote.url`. It is, however, recommended to leave this property empty so that no separate remote connector server is started, and instead to use the servlet container's remote connector server.

For more information about the management component see the [Blueprint Developer Manual](#).

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

- [Section 5.1, "General Concepts" \[78\]](#)
- [Section 5.2, "Monitoring Services" \[80\]](#)
- [Section 5.3, "JMX" \[85\]](#)
- [Section 5.4, "See Also" \[92\]](#)

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.

In order to configure available managed beans (MBeans) for JMX, components need to register them via `com.coremedia.jmx.MBeanRegistrar` as described in Section "The JMX Component" in *Blueprint Developer Manual*.

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” \[85\]](#).

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 6.3, “CAE Feeder JMX Managed Beans” in *Search Manual*.

The MBeans for CAE Feeder are by default configured via the artifact `com.coremedia.cms:cae-feeder-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

- ???

When monitoring by service, health checks for `WorkflowRepository` and `CapListRepository` are irrelevant.

- [Section 5.3.5, “Proactive Engine Sub Component” \[89\]](#)
- [Section 5.3.4, “Health \[Proactive Engine\]” \[89\]](#)

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.ContentBeanFactoryManager`
- `com.coremedia.objectserver.dataviews.AbstractDataViewFactoryManager`
- `com.coremedia.objectserver.view.resolver.TemplateViewRepositoryProviderManagement`
- `com.coremedia.objectserver.view.resolver.ViewResolverManagement`
- `com.coremedia.objectserver.web.links.LinkFormatterManager`

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" \[85\]](#)

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 6.2, “Content Feeder JMX Managed Beans” in *Search Manual*.

Recommended JMX Monitoring

- [Section 5.3.3, “Feeder” \[88\]](#)

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” \[86\]](#)

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” \[86\]](#)

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” \[86\]](#)
- [Section 5.3.6, “Replicator” \[90\]](#)

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” \[85\]](#)

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" \[85\]](#)

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" \[85\]](#)

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” \[80\]](#).

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” \[85\]](#) 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 <code>CapListRepositoryRequired</code> and <code>CapListRepositoryAvailable</code> (see JavaDoc for details). A value of <code>false</code> 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 <code>ContentRepositoryRequired</code> and <code>ContentRepositoryAvailable</code> (see JavaDoc for details). A value of <code>false</code> signals, that the required repository is unavailable.

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

Table 5.1. CapConnection JMX Monitoring

5.3.2 ContentServer

The attributes and metrics mentioned in [Table 5.2, “ContentServer JMX Monitoring” \[86\]](#) 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	<p>Time in epoch milliseconds when a license will expire and thus servers will fail to start. Note, that it is recommended to monitor <code>Server.LicenseValidUntilSoft</code> instead.</p>

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” [88] are suggested for monitoring.

For a complete overview of available attributes and metrics have a look at Section 6.2, “Content Feeder JMX Managed Beans” in *Search Manual*.

Feeder.State	
Type	Attribute
Value Type	String
Description	<p>The state of the Content Feeder which is one of the following:</p> <ul style="list-style-type: none"> • stopped • starting • initializing • running • failed <p>A typical health check monitors that the Content Feeder reaches state <i>running</i> after Content Feeder startup with a certain grace period.</p>
Feeder.StateNumeric	
Type	Attribute
Value Type	int
Description	<p>The state of the Content Feeder as number which is one of the following:</p> <ul style="list-style-type: none"> • 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” [89] 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	<p>Signals if the component is healthy in relation to the configuration.</p> <p>A typical health check monitors that the CAE Feeder reports that it is healthy after CAE Feeder startup with a certain grace period.</p>

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, “ProactiveEngine JMX Monitoring” [90] are suggested for monitoring.

For a complete overview of available attributes and metrics have a look at Section 6.3, “CAE Feeder JMX Managed Beans” in *Search Manual* and `com.core-`

`media.cap.persistentcache.proactive.ProactiveEngineManagement.`

ProactiveEngine.KeysCount	
Type	Gauge
Value Type	int
Description	<p>The total number of "keys" that need to be kept up-to-date by the CAE Feeder.</p> <p>The value may go down when destroying content or moving content outside the path configured for feeding.</p> <p>Should be monitored together with <code>ValuesCount</code>. See description of <code>ValuesCount</code> for details.</p>
ProactiveEngine.ValuesCount	
Type	Gauge
Value Type	int
Description	<p>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 <code>KeysCount</code> and thus <code>ValuesCount</code> is always less than or equal to <code>KeysCount</code>.</p> <p>The value may go down on invalidations.</p> <p>Monitoring typically observes the difference of <code>KeysCount</code> versus <code>ValuesCount</code>: A stable state is reached, when <code>KeysCount</code> is equal to <code>ValuesCount</code>. Otherwise, so if <code>ValuesCount</code> is less than <code>KeysCount</code>, the value of <code>ValuesCount</code> should increase over time. If it does not increase within a given amount of time, you should raise an alarm.</p>

Table 5.5. ProactiveEngine JMX Monitoring

5.3.6 Replicator

The attributes and metrics mentioned in [Table 5.6, "Replicator JMX Monitoring" \[91\]](#) 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*.

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

Table 5.6. Replicator JMX Monitoring

5.4 See Also

You will find additional documentation in the following sections:

- [Section 4.10, “JMX Management” \[76\]](#)
- Section “The JMX Component” in *Blueprint Developer Manual*
- 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 6.3, “CAE Feeder JMX Managed Beans” in *Search Manual*
- Section 6.2, “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.
CAE Feeder	Content applications often require search functionality not only for single content items but for content beans. The <i>CAE Feeder</i> makes content beans searchable by sending their data to the <i>Search Engine</i> , which adds it to the index.
Content Application Engine (CAE)	<p>The <i>Content Application Engine (CAE)</i> is a framework for developing content applications with <i>CoreMedia CMS</i>.</p> <p>While it focuses on web applications, the core frameworks remain usable in other environments such as standalone clients, portal containers or web service implementations.</p> <p>The CAE uses the Spring Framework for application setup and web request processing.</p>
Content Bean	A content bean defines a business oriented access layer to the content, that is managed in <i>CoreMedia CMS</i> and third-party systems. Technically, a content bean is a Java object that encapsulates access to any content, either to <i>CoreMedia CMS</i> content items or to any other kind of third-party systems. Various <i>CoreMedia</i> components like the <i>CAE Feeder</i> 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	<p>The <i>Content Delivery Environment</i> is the environment in which the content is delivered to the end-user.</p> <p>It may contain any of the following modules:</p> <ul style="list-style-type: none"> • <i>CoreMedia Master Live Server</i> • <i>CoreMedia Replication Live Server</i> • <i>CoreMedia Content Application Engine</i> • <i>CoreMedia Search Engine</i> • <i>Elastic Social</i> • <i>CoreMedia Adaptive Personalization</i>
Content Feeder	The <i>Content Feeder</i> is a separate web application that feeds content items of the <i>CoreMedia</i> repository into the <i>CoreMedia Search Engine</i> . Editors can use the <i>Search Engine</i> to make a full text search for these fed items.

Glossary |

Content item	In <i>CoreMedia CMS</i> , 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	<p>The <i>Content Management Environment</i> is the environment for editors. The content is not visible to the end user. It may consist of the following modules:</p> <ul style="list-style-type: none">• <i>CoreMedia Content Management Server</i>• <i>CoreMedia Workflow Server</i>• <i>CoreMedia Importer</i>• <i>CoreMedia Site Manager</i>• <i>CoreMedia Studio</i>• <i>CoreMedia Search Engine</i>• <i>CoreMedia Adaptive Personalization</i>• <i>CoreMedia Preview CAE</i>
Content Management Server	Server on which the content is edited. Edited content is published to the Master Live Server.
Content Repository	<i>CoreMedia CMS</i> 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	<p><i>Content Server</i> is the umbrella term for all servers that directly access the CoreMedia repository:</p> <p><i>Content Servers</i> are web applications running in a servlet container.</p> <ul style="list-style-type: none">• <i>Content Management Server</i>• <i>Master Live Server</i>• <i>Replication Live Server</i>
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 <i>CoreMedia CMS</i> . 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 .
Controm Room	<i>Controm Room</i> is a <i>Studio</i> plugin, which enables users to manage projects, work with workflows, and collaborate by sharing content with other <i>Studio</i> users.
CORBA (Common Object Request Broker Architecture)	<p>The term <i>CORBA</i> 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.</p> <p>CORBA programs communicate using the standard IIOP protocol.</p>

Glossary |

CoreMedia Studio	<p><i>CoreMedia Studio</i> 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.</p> <p>As a modern web application, <i>CoreMedia Studio</i> is based on the latest standards like Ajax and is therefore as easy to use as a normal desktop application.</p>
Dead Link	A link, whose target does not exist.
DTD	<p>A Document Type Definition is a formal context-free grammar for describing the structure of XML entities.</p> <p>The particular DTD of a given Entity can be deduced by looking at the document prolog:</p> <pre><!DOCTYPE coremedia SYSTEM "http://www.coremedia.com/dtd/coremedia.dtd"</pre> <p>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.</p>
Elastic Social	<p><i>CoreMedia Elastic Social</i> is a component of <i>CoreMedia CMS</i> that lets users engage with your website. It supports features like comments, rating, likings on your website. <i>Elastic Social</i> is integrated into <i>CoreMedia Studio</i> so editors can moderate user generated content from their common workplace. <i>Elastic Social</i> bases on NoSQL technology and offers nearly unlimited scalability.</p>
EXML	<p>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.</p>
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	<p>CoreMedia Headless Server is a CoreMedia component introduced with CoreMedia Content Cloud which allows access to CoreMedia content as JSON through a GraphQL endpoint.</p> <p>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.</p>
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 <i>Best current practice</i> (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,

Glossary |

	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, <i>Interoperable Object Reference</i> refers to the name with which a CORBA object can be referenced.
Jangaroo	<i>Jangaroo</i> is a JavaScript framework developed by CoreMedia that supports ActionScript as an input language which is compiled down to JavaScript. You will find detailed descriptions on the Jangaroo webpage http://www.jangaroo.net .
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 <i>Master Live Server</i> is the heart of the <i>Content Delivery Environment</i> . It receives the published content from the <i>Content Management Server</i> and makes it available to the <i>CAE</i> . If you are using the <i>CoreMedia Multi-Master Management Extension</i> you may use multiple <i>Master Live Server</i> 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. CoreMedia Studio uses the Open Source compiler Jangaroo 4 to translate MXML and ActionScript sources to JavaScript that is compatible with Ext JS 6.
Personalisation	On personalised websites, individual users have the possibility of making settings and adjustments which are saved for later visits.
Projects	A project is a collection of content items in CoreMedia CMS created by a specific user. A project can be managed as a unit, published or put in a workflow, for example.

Glossary |

Property	<p>In relation to CoreMedia, properties have two different meanings:</p> <p>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.</p> <p>In connection with the configuration of CoreMedia components, the system behavior of a component is determined by properties.</p>
Replication Live Server	<p>The aim of the <i>Replication Live Server</i> is to distribute load on different servers and to improve the robustness of the <i>Content Delivery Environment</i>. The <i>Replication Live Server</i> is a complete Content Server installation. Its content is an replicated image of the content of a <i>Master Live Server</i>. The <i>Replication Live Server</i> updates its database due to change events from the <i>Master Live Server</i>. You can connect an arbitrary number of <i>Replication Live Servers</i> to the <i>Master Live Server</i>.</p>
Resource	<p>A folder or a content item in the CoreMedia system.</p>
ResourceURI	<p>A ResourceUri uniquely identifies a page which has been or will be created by the <i>Active Delivery Server</i>. The ResourceUri consists of five components: Resource ID, Template ID, Version number, Property names and a number of key/value pairs as additional parameters.</p>
Responsive Design	<p>Responsive design is an approach to design a website that provides an optimal viewing experience on different devices, such as PC, tablet, mobile phone.</p>
Site	<p>A site is a cohesive collection of web pages in a single locale, sometimes referred to as localized site. In <i>CoreMedia CMS</i> a site especially consists of a site folder, a site indicator and a home page for a site.</p> <p>A typical site also has a master site it is derived from.</p>
Site Folder	<p>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.</p>
Site Indicator	<p>A site indicator is the central configuration object for a site. It is an instance of a special content type, most likely <code>CMSite</code>.</p>
Site Manager	<p>Swing component of CoreMedia for editing content items, managing users and workflows.</p> <p>The Site Manager is deprecated for editorial use.</p>
Site Manager Group	<p>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.</p>
Template	<p>In CoreMedia, JSPs used for displaying content are known as Templates.</p> <p>OR</p>

	<p>In <i>Blueprint</i> 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.</p>
Translation Manager Role	<p>Editors in the translation manager role are in charge of triggering translation workflows for sites.</p>
User Changes web application	<p>The <i>User Changes</i> web application is a <i>Content Repository</i> listener, which collects all content, modified by <i>Studio</i> users. This content can then be managed in the <i>Control Room</i>, as a part of projects and workflows.</p>
Version history	<p>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.</p>
Weak Links	<p>In general <i>CoreMedia CMS</i> always guarantees link consistency. But links can be declared with the <i>weak</i> attribute, so that they are not checked during publication or withdrawal.</p> <p>Caution! Weak links may cause dead links in the live environment.</p>
Workflow	<p>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.</p>
Workflow Server	<p>The <i>CoreMedia Workflow Server</i> 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.</p>
XLIFF	<p>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. <i>CoreMedia Studio</i> allows you to export content items in the XLIFF format and to import the files again after translation.</p>

Index

A

applications, 25
Architectural Overview, 14

C

communication
 between applications, 29
 encrypting CORBA, 34
 through firewall, 30
 using CORBA, 29
Communication of Components, 15
components, 57
configuration, 23, 55, 57
Control Room
 configuration, 43
 mongodb.client-uri, 43
 mongodb.prefix, 43
CORBA communication, 34
CoreMedia applications, 15
CoreMedia CMS, 1, 19, 22, 27
 directory structure, 27

D

directory structure, 27

F

firewall, 30

H

HTTPS, 39

J

Java, 21
JDK
 supported, 21

JMX management, 76
JPIF files, 25

L

licences
 IP-based, 45
 time-based, 45
license, 45
logback, 48
logging, 48
 applications, 48
 command-line tools, 49
 solr, 48

M

MBeans, 76
module.jpif, 25
monitoring, 77
 alert, **78**
 attribute, **78**
 counter, **78**
 gauge, **79**
 metric, **79**
 counter, **78**
 gauge, **79**

P

post-config.jpif, 25
pre-config.jpif, 25

S

security, 50
server utilities, 69-70, 73
single network interface, 33
Spring Boot
 HTTPS communication, 39
system requirements, 19

T

Third-Party Requirements, 18

U

User Changes web application
 configuration, 43

W

- Watchdog, 55
 - action elements, 58
 - component element, 57
 - databases, 56
 - edge element, 69
 - result codes, 73
 - starting, 56