CoreMedia Digital Experience Platform 8
//Version 7.5.45-10

# CoreMedia Operations Basics

# CoreMedia Operations Basics

# List of Figures

# List of Tables

# List of Examples

# 1. Introduction

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

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

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

- An overview of the architecture of the *CoreMedia CMS* system in Chapter 2, *Overview* [13]
- Some basics about the operating environments used by *CoreMedia Digital Experience Platform 8* are described in Chapter 3, *System Requirements* [17].
- the administration essentials of *CoreMedia CMS* for example how to start the applications, in Chapter 4, *Basics of Operation* [22]

## 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:

*Table 1.1. Typographic conventions*

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

In addition, these symbols can mark single paragraphs:

*Table 1.2. Pictographs*

| Pictograph | Description |
|---|---|
| ℹ | Tip: This denotes a best practice or a recommendation. |
| ⚠ | Warning: Please pay special attention to the text. |

| Pictograph | Description |
|---|---|
| ⚡ | Danger: The violation of these rules causes severe damage. |

## 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.

**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” [5] 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” [8] describes CoreMedia training. This includes the training calendar,the curriculum and certification information.
- Section 1.3.5, “CoreMedia Support” [9] 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” [9]) 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:

http://releases.coremedia.com/dxp8

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

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 CoreMedia Digital Experience Platform 8 Developer Manual.

### License files

You need license files to run the CoreMedia system. Contact the support (see Section 1.3.5, “CoreMedia Support” [9] ) 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:

http://documentation.coremedia.com/dxp8

The manuals have the following content and use cases:

*Table 1.3. CoreMedia manuals*

| Manual | Audience | Content |
|---|---|---|
| CoreMedia Utilized Open-Source Software | Developers, architects, administrators | This manual lists the third-party software used by CoreMedia and lists, when required, the licence texts. |
| 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. |
| Studio User Manual, English | Editors | This manual describes the usage of *CoreMedia Studio* for editorial and administrative work. It also describes the usage of the *Adaptive Personalization* and *Elastic Social* GUI that are integrated into *Studio*. |

| Manual | Audience | Content |
|---|---|---|
| LiveContext for IBM WebSphere Manual | Developers, architects, administrators | This manual gives an overview over the structure and features of CoreMedia LiveContext. It describes the integration with the IBM WebSphere Commerce system, the content type model, the *Studio* extensions, folder and user rights concept and many more details. It also describes administrative tasks for the features.<br><br>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. |
| 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. |
| Adaptive Personalization Manual | Developers, architects, administrators | This manual describes the configuration of and development with *Adaptive Personalization*, the CoreMedia module for personalized websites. You will learn how to configure the GUI used in *CoreMedia Studio*, how to use predefined contexts and how to develop your own extensions. |
| Analytics Connectors Manual | Developers, architects, administrators | This manual describes how you can connect your CoreMedia website with external analytic services, such as Google Analytics. |
| Content Application Developer Manual | Developers, architects | This manual describes concepts and development of the *Content Application Engine (CAE)*. You will learn how to write JSP or Freemarker templates that access the other CoreMedia modules and use the sophisticated caching mechanisms of the CAE. |
| Content Server Manual | Developers, architects, administrators | This manual describes the concepts and administration of the main CoreMedia component, the *Content Server*. You will learn about the content type model which lies at the heart of a CoreMedia system, about user and rights management, database configuration, and more. |

| Manual | Audience | Content |
|---|---|---|
| Elastic Social Manual | Developers, architects, administrators | This manual describes the concepts and administration of the *Elastic Social* module and how you can integrate it into your websites. |
| Importer Manual | Developers, architects | This manual describes the structure of the internal CoreMedia XML format used for storing data, how you set up an *Importer* application and how you define the transformations that convert your content into CoreMedia content. |
| Search Manual | Developers, architects, administrators | This manual describes the configuration and customization of the *CoreMedia Search Engine* and the two feeder applications: the *Content Feeder* and the *CAE Feeder*. |
| Site Manager Developer Manual | Developers, architects, administrators | This manual describes the configuration and customization of *Site Manager*, the Java based stand-alone application for administrative tasks. You will learn how to configure the *Site Manager* with property files and XML files and how to develop your own extensions using the *Site Manager API*. |
| Studio Developer Manual | Developers, architects | This manual describes the concepts and extension of *CoreMedia Studio*. You will learn about the underlying concepts, how to use the development environment and how to customize *Studio* to your needs. |
| Unified API Developer Manual | Developers, architects | This manual describes the concepts and usage of the *CoreMedia Unified API*, which is the recommended API for most applications. This includes access to the content repository, the workflow repository and the user repository. |
| Workflow Manual | Developers, architects, administrators | This manual describes the *Workflow Server*. This includes the administration of the server, the development of workflows using the XML language and the development of extensions. |

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

*Support request*

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

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

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

*Support checklist*

To put it in a nutshell, CoreMedia needs:

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

*Log files*

An essential feature for the CoreMedia system administration is the output log of Java processes and CoreMedia components. They're often the only source of information for error tracking and solving. All protocolling services should run at the highest log level that is possible in the system context. For a fast breakdown, you should be logging at debug level. The location where component log output is written is specified in its < `appName>-logback.xml` file.

**Which Log File?**

Mostly at least two CoreMedia components are involved in errors. In most cases, the *Content Server* log files in `coremedia.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 the "Logging" chapter of the [Operations Basics Manual] for details.

*Table 1.4. Log files check list*

| Component | Problem | Log files |
|---|---|---|
| CoreMedia Studio | general | `CoreMedia-Studio.log` `coremedia.log` |
| CoreMedia Editor | general | `editor.log` `coremedia.log` `workflowserver.log` `capclient.properties` |
| | check-in/check-out | `editor.log` `coremedia.log` `workflowserver.log` `capclient.properties` |
| | publication or preview | `coremedia.log` (*Content Management Server*) `coremedia.log` (Master Live Server) |

| Component | Problem | Log files |
|---|---|---|
| | | `workflowserver.log`<br>`capclient.properties` |
| | import | `importer.log`<br>`coremedia.log`<br>`capclient.properties` |
| | workflow | `editor.log`<br>`workflow.log`<br>`coremedia.log`<br>`capclient.properties` |
| | spell check | `editor.log`<br>`MS Office version details`<br>`coremedia.log` |
| | licenses | `coremedia.log`<br>(*Content Management Server*)<br>`coremedia.log`<br>(Master Live Server) |
| Server and client | communication errors | `editor.log`<br>`coremedia.log`<br>(*Content Management Server*)<br>`coremedia.log`<br>(Master Live Server)<br>`*.jpif files` |
| | preview not running | `coremedia.log` (content server)<br>`preview.log` |
| | website not running | `coremedia.log`<br>(*Content Management Server*)<br>`coremedia.log`<br>(Master Live Server)<br>*coremedia.log*<br>(*Replication Live Server*)<br>`Blueprint.log`<br>`capclient.properties`<br>`license.zip` |
| Server | not starting | `coremedia.log`<br>(*Content Management Server*)<br>`coremedia.log`<br>(Master Live Server)<br>`coremedia.log`<br>(*Replication Live Server*)<br>`capclient.properties`<br>`license.zip` |

## 1.4 Change Chapter

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

*Table 1.5. Changes*

| Section | Version | Description |
|---|---|---|
| | | |
| Section 4.5.2, “Deployment” [44] | 7.5.41 | Removed properties `controlroom.jdbc.driver`, `controlroom.jdbc.url`, `controlroom.jdbc.user` and `controlroom.jdbc.password`. *Control Room* no longer supports *IBM DB2* for persisting collaboration data. See [CoreMedia DXP 8 Manual], Section "In-Memory Replacement for MongoDB-Based Services". |

# 2. Overview

*Overview and deployment*

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

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 4.2, “Communication between the System Applications” [24] for details. Table 2.1, “CoreMedia applications” [13] shows all applications of *CoreMedia CMS*, describes what they do, if there are multiple instances and with which applications they communicate:

*Table 2.1. CoreMedia applications*

| Application | Purpose | Multiple Instances | Communicates with |
|---|---|---|---|
| Content Management Server | Manages the content in the Management Environment and publishes content to the *Master Live Server*. | No | → All clients<br>→ Publishes content to the Master Live Server<br>→ External relational database<br>→ Search Engine |
| Master Live Server | Manages the content in the Delivery Environment | Multiple instances when Multi-Site is used | → All clients.<br>→ External relational database |
| Replication Live Server | Serves content to the CAEs | Multiple instances can be attached to one *Master Live Server* | → Content Application Engine<br>→ External relational database |
| Workflow Server | Executes workflows | No | Content Management Server |

| Application | Purpose | Multiple Instances | Communicates with |
|---|---|---|---|
| Site Manager | Management tool for workflows and users. | Yes | → Content Management Server<br>→ Workflow Server |
| Studio | Content editing and management. Hosts management extensions for *Elastic Social* and *Adaptive Personalization*. | One web application | → Content Management Server<br>→ Search Engine<br>→ Workflow Server<br>→ MongoDB |
| Search Engine | Indexes content and provides searches functionality. | Yes. | → Content Management Server<br>→ Content Feeder<br>→ CAE Feeder<br>→ Studio<br>→ Content Application Engine |
| CAE Feeder | Feeds content beans into the *Search Engine* | Multiple instances possible, for example when reindexing. | → Content Management Server<br>→ Search Engine<br>→ External relational database |
| Content Feeder | Serves content to the *Search Engine* | Multiple instances possible, for example when reindexing. | → Content Management Server<br>→ Search Engine |
| Content Application Engine | Serves sites to the customer. Hosts *Elastic Social* and *Adaptive Personalization* extension. | Multiple instances can be attached to one *Master Live Server* or *Replication Live Server* | → Content Server<br>→ MongoDB database for *Elastic Social*<br>→ Search Engine<br>→ Custom external systems |

| Application | Purpose | Multiple Instances | Communicates with |
|---|---|---|---|
| Importer | Imports content into the *Content Management Server*. | Yes | ⟶ Content Management Server |
| Watchdog | Monitors CoreMedia applications and databases | Yes | Monitored applications |

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

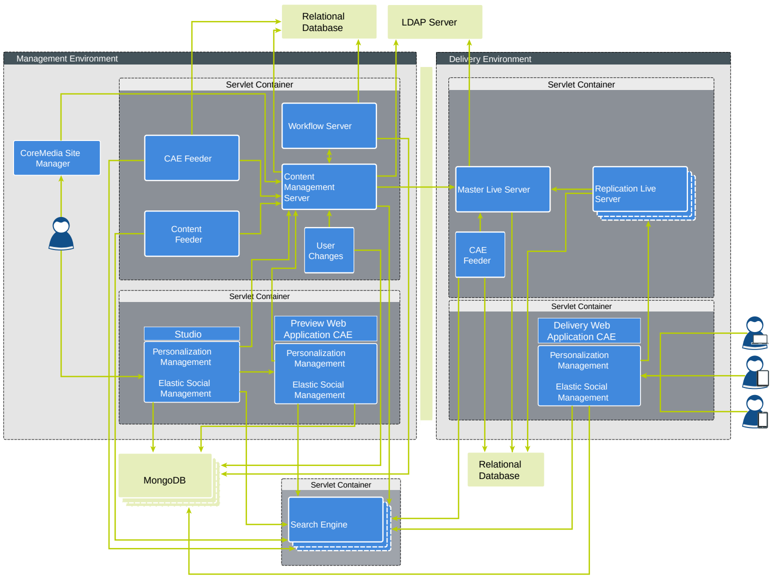

*Figure 2.1. Architectural Overview*

*Third-party software*

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

- A Java installation.
- A relational database for the content storage.
- A servlet container as a runtime environment for most of the applications.
- A MongoDB database for *Elastic Social*.

- A browser for *CoreMedia Studio*.

In addition, you can run *CoreMedia CMS* with an LDAP server. Find a list of all supported environments at the CoreMedia Online Documentation.

*Installation*

*CoreMedia CMS* is not shipped with an installer. Instead, it comes as a development workspace, where you can customize it to your needs. By default, this workspace produces RPM and ZIP artifacts that you can install as usual. See CoreMedia Digital Experience Platform 8 Developer Manual for more details about the workspace.

*Security*

The communication between all applications can be secured. See Section 4.2, “Communication between the System Applications” [24] for details.

*Logging and Monitoring*

All applications of *CoreMedia CMS* use Logback for logging. See Section 4.7, “Logging” [50] 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" of the 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.

# 3. System Requirements

*Use only recommended systems*

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

- *Certified level*

  Certified infrastructure components are extensively tested to work with the *CoreMedia CMS* system. Every infrastructure component approved with the first final CMS Release is certified. It is recommended to use these components for productive systems.
- *Supported level*

  Supported infrastructure components will also work with CoreMedia applications but they are tested less exhaustively, because they are released after the first final CMS Release. They also can be used for productive systems. Refer to the `notes.html` file for announcements of additionally supported environments or the appendix 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.

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

*Supported operation environments*

You will find the approved components in the Supported Operation Environments document on the documentation page. In the following sections you will find some general hints for the usage of these components:

- Java platforms in Section 3.1, “Java” [19],
- Databases in Section 3.2, “Databases” [20],
- Servlet containers in Section 3.3, “Servlet Containers” [21].

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).

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/).

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*.

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 https://documentation.coremedia.com/dxp8/supported-environments-en.pdf.

## 3.3 Servlet Containers

*CoreMedia CMS* web applications such as the *Content Servers* and *Content Application Engine* use a servlet container for operation. Default servlet container is Tomcat. Keep in mind, that the servlet containers have only been tested in *CoreMedia* compliant operating environments and therefore are only approved on these platforms. For all supported servlet containers see the [Supported Environments] document at https://documentation.coremedia.com/dxp8/supported-environments-en.pdf.

# 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 starting and operating CoreMedia applications.

## 4.1 CoreMedia CMS Directory Structure

CoreMedia applications come as simple applications or as web applications. Both have a standard directory structure that will be described in the following. Other - application specific - directories, are described in the corresponding application manual.

### CoreMedia applications

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

- `./bin`: Start scripts (see Section 4.3, “Starting the Applications” [39]) 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” [50]).
- `./var/run`: runtime data (such as Process ID).
- `./var/tmp`: temporary data.

### CoreMedia web application

A CoreMedia web application, the *Content Application Engine* or the *Content Server* for example, is deployed in a servlet container. The web applications have some of the following sub directories in its `WEB-INF` directory:

- `./config/<application>`: XML configuration files of the application.
- `./properties/corem`: *CoreMedia CMS* configuration files in Java properties format.
- `./lib`: Runtime resources like Java JAR files and DLLs.
- `./spring`: Spring configuration files of the web application.

## 4.2 Communication between the System Applications

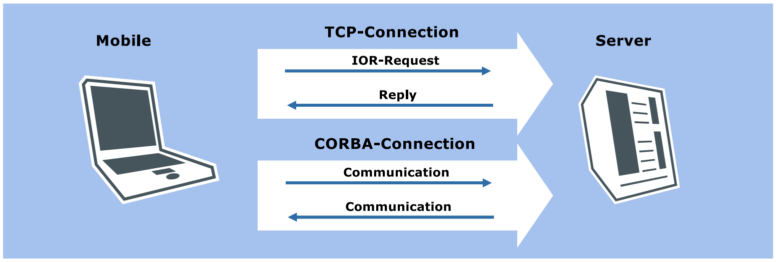

*Figure 4.1. IOR inquiry and answer between CoreMedia Client and Server*

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>/coremedia/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. Both values are defined in the `contentserver.properties` file.
Example: The *Content Server* host has the name productionserver and the `contentserver.properties` file contains the property `cap.server.http.port=44445`. In this case, you can obtain the IOR with the following URL:
`cap.client.server.ior.url=http://productionserver:44445/coremedia/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 ORB property `com.sun.CORBA.ORBServerHost`. In the following example, the ORB is configured to embed its numeric address, by setting a system property:

```
-Dcom.sun.CORBA.ORBServerHost="10.1.3.253"
```

All CoreMedia applications deployed as web applications expect an ORB to be provided by the application server. To use the application server ORB you have to provide the JNDI name of the ORB in the property `com.coremedia.orb.jndiName`, for example:

```
com.coremedia.orb.jndiName=java:comp/ORB
```

If this property is left empty, each CoreMedia application will create its own ORB.

For Tomcat deployment, CoreMedia provides an integration of the Oracle JDK ORB. Take a look at the `tomcat-config` module in the *CoreMedia Blueprint* workspace for further details. All system properties defined, for example, in Tomcat's `setenv.sh` / `setenv.bat` are passed on as configuration properties to the ORB.

For WebSphere deployment, CoreMedia provides an integration of the IBM JDK ORB (see CoreMedia IBM Deployment Manual for IBM specific deployment).

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.ior.url` of the file `publisher.properties`.
- The *Replication Live Server* (when installed) has to communicate with its *Master Live Server*.
  The IOR URL is stored in the property `replicator.publicationIorUrl` of the file `replicator.properties`.

## 4.2.1 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 ORB property `com.sun.CORBA.ORBServerPort`. In the following example, the ORB is configured to listen on port 55555, by setting a system property:

- `-Dcom.sun.CORBA.ORBServerPort=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” [26] shows the scenario:

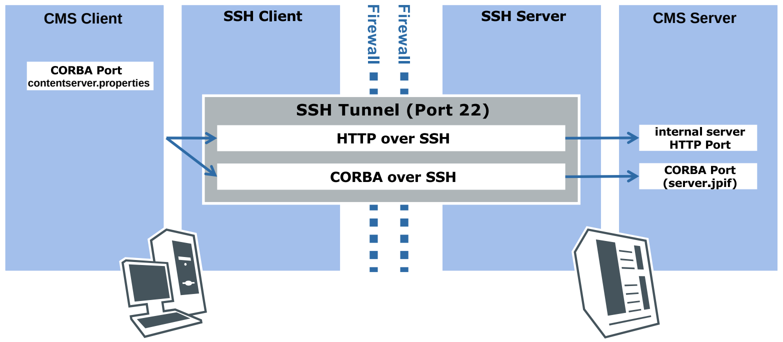

*Figure 4.2. Schema of the SSH tunnel*

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,

- 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 in the `contentserver.properties` files.
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>:<HTTP
   Port>/coremedia/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.

   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 ORB properties and system properties.

   Depending on the type of client, system properties are set either in the JPIF file (for command line tools), in the JNLP file (for Web Start) or in `setenv.sh/setenv.bat` (for web applications deployed in Tomcat). ORB properties are also set as system properties, except for command line tools, described below.

   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:

Table 4.1. Properties for SSH configuration

| Property Type | Property Name | Property Value |
|---|---|---|
| ORB | `com.sun.CORBA.legacy.connection.ORBSocketFactoryClass` | com.coremedia.corba.ORBRedirector50 |
| System | `com.coremedia.corba.ORBRedirector.original.host` | <CMSServer> |
| System | `com.coremedia.corba.ORBRedirector.redirect.host` | <SSHClient> |
| System | `com.coremedia.corba.ORBRedirector.original.port` | <CorbaPort> |
| System | `com.coremedia.corba.ORBRedirector.redirect.port` | <CorbaPort> |

7. In order to instruct stand-alone *Unified API* clients like the command line tools to contact <SSHClient> instead of <CMSServer>, you must configure the ORB property for the socket factory in the connection parameters for the *Unified API*.

   If you are setting up the *Unified API* connection programmatically, consider using the `connect(Map)` method of the class `Cap`.

```
Map parameters = new HashMap();
...
parameters.put
("com.sun.CORBA.legacy.connection.ORBSocketFactoryClass",
  "com.coremedia.corba.ORBRedirector50");
connection = Cap.connect(parameters);
```

   In any case, you may inject the parameter through the IOR URL passed to the *Unified API*. For command line tools, you can pass the URL on the command line:

```
cm systeminfo -url http://<SSHCLIENT>:<HTTPPort>/coremedia/ior?
  com.sun.CORBA.legacy.connection.ORBSocketFactoryClass=
  com.coremedia.corba.ORBRedirector50 -u admin -p admin
```

   You can also set the extended URL in the file `capclient.properties`:

```
cap.client.server.ior.url=\
  http://<SSHCLIENT>:<HTTPPort>/coremedia/ior?\
  com.sun.CORBA.legacy.connection.ORBSocketFactoryClass=\
  com.coremedia.corba.ORBRedirector50
```

It is also possible to pass an extended URL when opening a connection programmatically.

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

The `ORBRedirector` only works if the client uses the ORB from the Oracle J2RE. It may not work if you are not using an Oracle J2RE or an application runs in a third-party web container that provides its own ORB.

## 4.2.2 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 adding an `address` attribute to the Tomcat's `Connector` element (see http://tomcat.apache.org/tomcat-7.0-doc/config/http.html ).

For limiting the access through CORBA, too, some properties must be set. By setting `com.sun.CORBA.ORBServerHost` to the correct IP address, you ensure that external clients contact the server through the correct interface. In order to bind only the correct interface, you must configure a custom CoreMedia socket factory, which is configured using a system property. Set the following system properties when starting the *Content Management Server* and the *Workflow Server*:

*Table 4.2. Properties for Single IP configuration*

| Property Type | Property Name | Property Value |
|---|---|---|
| ORB | `com.sun.CORBA.ORBServerHost` | <IpAddress> |
| ORB | `com.sun.CORBA.legacy.connection.ORBSocketFactoryClass` | com.coremedia.corba.SingleIpSocketFactory50 |
| System | `com.coremedia.corba.SingleIpSocketFactory.ip` | <IpAddress> |

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, you have to use different factories, see Section 4.2.4, “Encrypting CORBA with SSL and bind to a Single Network Interface” [36] for details.

## 4.2.3 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.

In the example, the following ports numbers are used. You may want to use different port numbers for your deployment.

*Table 4.3. Example SSL Ports*

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

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* and *Workflow Server*.
2. Prepare the *Content Server* for SSL communication

3. Prepare the *Workflow Server* for SSL communication
4. Prepare the *Site Manager* 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 system properties to the content server's `setenv.sh/setenv.bat` files:

| Property Type | Property Name | Property Value |
|---|---|---|
| ORB | `com.sun.CORBA.ORBServerPort` | 14300 |
| ORB | `com.sun.CORBA.legacy.connection.ORBSocketFactoryClass` | com.coremedia.corba.SSLClientServerSocketFactory50 |
| ORB | `com.sun.CORBA.transport.ORBListenSocket` | SSL:14443 |
| System | `com.coremedia.corba.SSLServerSocketFactory.keystore` | <path to contentserver.keystore> |

*Table 4.4. Properties for Content Server SSL configuration*

| Property Type | Property Name | Property Value |
|---|---|---|
| System | com.core-media.corba.SSLServer-SocketFactory.passphrase | <mypassword> |

2. Place the `contentserver.keystore` in the folder `etc/keys/` of your installation home directory of the CMS server. For another location adjust the key store setting by defining the corresponding system property accordingly.

## Prepare the *Workflow Server* for SSL communication

1. Add the following two system properties during the invocation of the server:

*Table 4.5. Properties for Workflow Server SSL configuration*

| Property Type | Property Name | Property Value |
|---|---|---|
| ORB | com.sun.CORBA.ORBServer-Port | 14305 |
| ORB | com.sun.CORBA.legacy.con-nection.ORBSocketFactory-Class | com.coremedia.corba.SSLClient-ServerSocketFactory50 |
| ORB | com.sun.CORBA.trans-port.ORBListenSocket | SSL:14445 |
| System | com.core-media.corba.SSLServer-SocketFactory.keystore | <path to workflowserver.keystore> |
| System | com.core-media.corba.SSLServer-SocketFactory.passphrase | <mypassword> |

2. Place the `workflowserver.keystore` in folder `etc/keys/` of your installation home directory of the *Workflow Server*. For another location adjust the key store setting by defining the corresponding system property accordingly.

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

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

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

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

| Property Type | Property Name | Property Value |
|---|---|---|
| System | `com.coremedia.corba.SSLClientSocketFactory.clearTextPort` | 14300 |
| System | `com.coremedia.corba.SSLClientSocketFactory.sslPort` | 14443 |
| System | `com.coremedia.corba.SSLClientSocketFactory.keystore` | <path to workflowserver.keystore> |
| System | `com.coremedia.corba.SSLClientSocketFactory.passphrase` | <mypassword> |

*Table 4.6. Properties for Workflow to Content Server SSL configuration*

### Prepare the *Site Manager* for SSL communication

1. 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 `tomcat/webapps/editor-webstart/webstart/editor.jnlp`, behind the property tag with `name="java.security.policy"` inside the resources tag.

```
<property name=
 "com.sun.CORBA.legacy.connection.ORBSocketFactoryClass"
  value="com.coremedia.corba.SSLClientSocketFactory50"/>
<property
  name="com.coremedia.corba.SSLClientSocketFactory.
  clearTextPort"
value="14300,14305"/>
<property name=
 "com.coremedia.corba.SSLClientSocketFactory.sslPort"
  value="14443,14445"/>
```

```
<property
  name="com.coremedia.corba.SSLClientSocketFactory.keystore"
  value="
$$codebaseproperties/corem/editor.keystore"/>
<property name=
 "com.coremedia.corba.SSLClientSocketFactory.passphrase"
value="mypassword"/>
```

3. Place the `editor.keystore` in `tomcat/webapps/editor-webstart/properties/corem/` of your installation.

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

## Preparing a client ORB for SSL communication

1. CoreMedia clients running as web applications, such as the *Content Application Engine*, are usually configured to use the servlet container's ORB. CoreMedia provides an integration of the Oracle ORB into Tomcat. You can configure this ORB for SSL by setting the following system properties:

| Property Type | Property Name | Property Value |
|---|---|---|
| ORB | `com.sun.CORBA.legacy.connection.ORBSocketFactoryClass` | com.coremedia.corba.SSLClientSocketFactory50 |
| System | `com.coremedia.corba.SSLClientSocketFactory.clearTextPort` | 14300,14305 |
| System | `com.coremedia.corba.SSLClientSocketFactory.sslPort` | 14443,14445 |
| System | `com.coremedia.corba.SSLClientSocketFactory.keystore` | <path to workflowserver.keystore> |

*Table 4.7. Properties for Client ORB SSL configuration*

| Property Type | Property Name | Property Value |
|---|---|---|
| System | `com.core-media.corba.SSLClient-SocketFactory.passphrase` | <mypassword> |

## Prepare *Unified API* clients for SSL communication

1. In order to instruct stand-alone *Unified API* clients like the command line tools to use SSL, the ORB properties must be set in the connection parameters for the *Unified API* instead of as system properties.
2. If you are setting up the *Unified API* connection programmatically, consider using the `connect(Map)` method of the class `Cap`.

```
Map parameters = new HashMap();
...
parameters.
put("com.sun.CORBA.legacy.connection.ORBSocketFactoryClass",
  "com.coremedia.corba.SSLClientSocketFactory50");
connection = Cap.connect(parameters);
```

3. In any case, you may inject the parameter through the IOR URL passed to the *Unified API*. For command line tools, you can pass the URL on the command line:

```
cm systeminfo
  -url http://<SSHCLIENT>:<CorbaPort>/coremedia/ior?
  com.sun.CORBA.legacy.connection.ORBSocketFactoryClass=
  com.coremedia.corba.SSLClientSocketFactory50
  -u admin -p admin
```

→ You can also set the extended URL in the file `capclient.properties`:

```
cap.client.server.ior.url=\
  http://<SSHCLIENT>:<CorbaPort>/coremedia/ior?\
  com.sun.CORBA.legacy.connection.ORBSocketFactoryClass=\
  com.coremedia.corba.SSLClientSocketFactory50
```

→ It is also possible to pass an extended URL when opening a connection programmatically.

## 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.

### 4.2.4 Encrypting CORBA with SSL and bind to a Single Network Interface

It is also possible to bind to a single network interface and to encrypt the CORBA communication with SSL.

Limiting the access of the HTTP connection to a single IP and encrypting it can be achieved by combining the settings described in the previous sections.

For limiting the encrypted CORBA connection to a single network interface you should first configure your system for the SSL encryption as shown in the last section, ensure that everything works and then set the following system properties for the *Content Server* and the *Workflow Server*:

```
-Dcom.sun.CORBA.legacy.connection.ORBSocketFactoryClass=
com.coremedia.corba.SingleIpSSLClientServerSocketFactory50

-Dcom.coremedia.corba.SingleIpSSLServerSocketFactory.ip=<IpAddress>
```

Replace `<IpAddress>` by the IP address of the network interface to bind, for example 10.1.3.253.

### 4.2.5 Preparing Tomcat 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 (*WebDAV*, *CAE*) connection via HTTPS. This chapter describes how you create a key and how you configure Tomcat to use this key. If you want a more detailed description, please visit the Tomcat documentation at http://tomcat.apache.org.

## Creating a Key

In order to connect client and servlet container via HTTPS you have to generate a key for the servlet container. This key is sent from servlet container 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 tomcat
-keystore /example/coremedia/.keystore -keyalg rsa
```

   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 tomcat` ). A key is created according to the RSA algorithm. The key is saved in the `keystore` file `/example/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 < Tomcat>".

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 Tomcat

In addition to the creation of a key for HTTPS communication some entries must be made in certain configuration files. `<ServletContainerHome>` stands here for the home directory of the servlet container. The Tomcat servlet container that is part of the *CoreMedia Project* deployment workspace already contains the following entries. If you use the deployment workspace, you can simply configure the required settings in the `catalina.properties` file in the `tomcat-template` module.

#### Entry in `<TomcatHome>/conf/server.xml`

In the file `<ServletContainerHome>/conf/server.xml` you must insert the following section after the already existing `<Connector>` elements:

*Example 4.1. Code to insert into server.xml*

```
<!-- Secure HTTP -->
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
  maxThreads="150"   enableLookups="false"
  disableUploadTimeout="true"  acceptCount="100" scheme="https"
  secure="true"  clientAuth="false" sslProtocol="TLS"
  keystoreFile="/example/coremedia/.keystore"
  keystorePass="changeit"/>
```

Adjust the entries `port` (port number with which the browser communicates), `keystoreFile` and `keystorePass` to your own settings. The key required for HTTPS communication is stored in the key store file generated in Section “Creating a Key” [37]. The path to the key store given there must match the path you entered in `server.xml`.

## 4.2.6 Troubleshooting

*You use Oracle's ORB implementation. 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.transport.ORBUseNIOSelectToWait=false
```

## 4.3 Starting the Applications

In *CoreMedia* there are two distinct types of applications: *web applications* and *command-line tools*.

Examples for web applications:

- the *Content Servers*
- *Workflow Server*
- *CAE Feeder*

Examples for command-line tools:

- *runlevel*
- *serverimport*
- *schemaaccess*

General preconditions for starting *CoreMedia CMS* applications:

- An approved browser must be installed in order to use *CoreMedia Studio*.
- For operation of *CoreMedia CMS* command line tools on Unix systems, the ksh shell must be installed in `/bin`. It is used for starting the CoreMedia applications on execution of the `cm` command (see next section).
- In general, it must be ensured that all computers in use are mutually visible over DNS.
- The image converter service of the server needs the program convert from ImageMagick. On a Linux and Solaris system, you have to use the convert of your distribution. If there is no `convert` installed, or if you are using Windows you can download the respective program at http://www.imagemagick.org. You have to enter the path to the installed convert in the property `convertCommand` in the file `properties/corem/imageconverter.properties`. You can find a description of the other image converter properties in the description of the `AggregatingImageBlobEditor` in the *Site Manager Developer Manual*.

### 4.3.1 Starting CoreMedia Web Applications

The *CoreMedia* web applications are deployed as web applications in a supported servlet container. These applications are started together with the servlet container. Therefore, refer to the servlet container's documentation for how to set JVM options and system properties.

## 4.3.2 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 (in the example: `cms-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 the [Content Server Manual] and [Workflow Server 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]).

*Example 4.2. Output of `cm` in the `cms-tools` directory*

```
$ 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
```

### Configuration of the Start Routine with JPIF Files

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

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

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

The memory usage within the Java Virtual Machine can be configured using the parameters `-Xms<size>` and `-Xmx<size>`. `-Xms` specifies the initial object memory size and `-Xmx` the maximum object memory size. The memory requirement for the applications is preconfigured and depends on the main memory size according to the standard hardware recommendations, but can be increased using this parameter if necessary.

The ORB can be configured to use a fixed CORBA port using the parameter `com.sun.CORBA.ORBServerPort` as described in Section 4.2, “Communication between the System Applications” [24].

Furthermore, the target of the log outputs of the Java process (see Section 4.7, “Logging” [50]) can be configured with the parameter `OUTPUT_REDIRECT`.

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

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

In general, these files need not be changed.

## 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`.

## 4.4 Configuration of CoreMedia Applications

*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):

Web applications:

- `WEB-INF`
- `WEB-INF/properties/corem`
- `WEB-INF/config`

Command-line tools:

- `properties/corem`
- `config`

The manuals of the applications contain descriptions of how to use the relevant property file(s) to configure each CM application.

Most web applications also offer the possibility to configure properties via JNDI, so that you can leave the WAR files untouched and for example define properties in the `context.xml` of the Tomcat installation. For details see the CoreMedia Digital Experience Platform 8 Developer Manual.

Besides directly editing the property files configuration might also be done at Runtime via JMX. For details about what can be configured please consult the manuals of the applications. For more general details about JMX see Section 4.9, “JMX Management” [71].

**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.

## 4.5 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 Mongo DB database. When deployed with a MongoDB database, configure the collaborative components to connect to your MongoDB instance using the configuration properties given below.

*Table 4.8. Properties for persistence of collaboration data to MongoDB*

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

## 4.5.3 Recovery of Collaborative Components Database

In this chapter you will get to know how to backup 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 collborative components database.
2. Backup the *Content Management Server*'s database.

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 CoreMedia Content Server Manual for information how to backup the Content Server's database.

You can find an overview about backup of Mongo DB 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* web application.
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* web application.

## 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` in the `contentserver.properties` file 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 *CoreMedia Content Server Manual*) to get this information and the utility `usedlicenses` (see Section “Usedlicenses” in *CoreMedia 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` in the file `contentserver.properties` defines the location of the license file relative to the `WEB-INF` directory of the server web application. 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:**

*Example 4.3. A sample license file*

```
<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="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="filesystem" concurrent-users="5"
   named-users="50"/>
  <License service="replicator" concurrent-users="5"
   named-users="10"/>
  <License service="feeder" concurrent-users="2"
   named-users="10"/>
  <License service="analytics" concurrent-users="2"
   named-users="10"/>
  <License service="dashboard" concurrent-users="30"
   named-users="50"/>
</LicenseConfiguration>
```

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

*Table 4.9. Elements of a license file*

| Element | Attribute | Description |
|---|---|---|
| Server | type | The type of the server for which the license is valid. Possible values are:<br>➞ *production*: The *Content Management Server*<br>➞ *publication*: The Master Live Server<br>➞ *live*: The *Replication Live Server* |
| Property | name | The aim of the property. Possible values are:<br>➞ licensed-to: The customer to which the system is licensed.<br>➞ workflow: Defines if the programmable workflow is licensed ("enabled").<br>➞ analytics: Defines if Analytics is licensed ("enabled").<br>➞ elastic-social: Defines if Elastic Social is licensed ("enabled").<br>➞ personalization: Defines if Adaptive Personalization is licensed ("enabled").<br>➞ elastic-social: Defines if Elastic Social is licensed ("enabled").<br>➞ 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. |

| Element | Attribute | Description |
|---|---|---|
| | `multiplicity` | The maximum number of sessions a user is allowed to open. |

## 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 DXP 8* 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 the Apache Solr web application, which uses Apache Log4J.

### 4.7.1 Logging Configuration for Web Applications

*CoreMedia* web applications use a common logging component which is based on Logback. The log configuration for each web application is located in `WEB-INF/logback.xml`. If no configuration file is provided, a fallback configuration is used.

The default log directory is set to `./logs`. It can be changed by providing the property `coremedia.logging.directory` either in `WEB-INF/application.properties`, as a system property, or as a JNDI property `java:comp/env/coremedia/logging/directory`.

It is also possible to access the log configuration via JMX using `com.coremedia:Type=Logging,application=<applicationname>`.

The logging configuration contains reasonable defaults which you can override if required. You can define the properties `log.pattern` and `log.file` in your project `logback.xml` file. For example to enable tenant logging for *Elastic Social* in your log pattern it can be defined like this:

```
 <property name="log.pattern" value="%d{yyyy-MM-dd HH:mm:ss}
%-7([%level]) %logger - [%X{tenant}] %message \\(%thread\\)%n" />
```

For more information about the logging component, see the Section "The Logging Component" in *CoreMedia Digital Experience Platform 8 Developer Manual* for details.

### 4.7.2 Logging Configuration for Apache Solr

The Apache Solr web application uses Apache Log4J as log framework, which is configured in the file `log4j.properties`.

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://cwiki.apache.org/confluence/display/solr/Configuring+Logging for details on Solr log configuration.

### 4.7.3 Logging Configuration for Command-Line Tools

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

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

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

**`stdout/stderr` Output**

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

## 4.8 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” [54] for a detailed description of the actions.

| Application | Action |
|---|---|
| Content Server | → `<ServerQuery>`: Checks the overall status<br>→ `<ServiceStatus>`: Checks the status of a specific service<br>→ `<ServerMode>`: Checks the runlevel |
| Workflow Server | → `<WorkflowServerQuery>`: Checks the overall status of a *Workflow Server*<br>→ `<ProcessStatus>`: Checks if a specified number of processes is running |
| Database | → `<DB>`: Checks the status of the database |
| CAE | → `<Http>`: 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*

## 4.8.1 Starting the Watchdog/Probedog

### Watchdog

The watchdog is always started with the servlet container.

### Probedog

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

## 4.8.2 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 Managment Server*.

## 4.8.3 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” [65] 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:

*Table 4.11. Attributes of the Component element*

| Attribute | Optional | Description |
|---|---|---|
| `name` | | A unique name. |
| `startAction` | | 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. |
| `delay` | x | The attribute specifies the delay in seconds until the action defined in the `startAction` attribute is executed. |

### 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:

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

*Table 4.12. General attributes of the Action element*

| 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. |

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

Possible actions

*Table 4.13. Custom Action*

| **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.<br><br>Example:<br><br>`<Custom name="my first action" timeout="10">`<br>` <Action class="com.customer.MyWatchDogAction" myprop="ok">`<br>`   <MySub class="com.customer.MySub" foo="bar"/>`<br>`   <MySub class="com.customer.MySub" foo="baz"/>`<br>` </Action>`<br>`</Custom>`<br><br>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 al- |

| **Custom** | |
|---|---|
| | lows for elaborate configuration. Please read the chapter "The BeanParser" of the *Workflow Manual* for details.<br><br>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.<br><br>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 4.14. Action DB*

| **DB** | |
|---|---|
| Description | This action checks a CoreMedia database over the JDBC API. |
| Attribute | *`driver`*<br><br>JDBC driver to use<br><br>*`url`*<br><br>URL where to connect to the database<br><br>*`user`*<br><br>Name of the database user<br><br>*`password`*<br><br>Password of the database user<br><br>*`propertyFile`*<br><br>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 Chapter 1, *Introduction* in *CoreMedia Content Server Manual* for details of the syntax of the configuration file.<br><br>You have to specify either driver, URL, user and password or the *`propertyFile`* attribute.<br><br>*`sql`* |

| DB | |
|---|---|
| | The database statement that is used to check the database. Default is `SELECT * FROM Resources WHERE id_ = 1`. |
| Error codes | 0 (ok)<br>10 (unexpected_error)<br>11 (timeout)<br>12 (error)<br>13 (respawning_too_fast)<br>21 (io_error)<br>61 (no_jdbc_driver)<br>62 (no_connection)<br>63 (sql_exception) |

*Table 4.15. Action HTTP*

| 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`*<br>Requested URL<br>*`user`*<br>User name to use for HTTP basic authentication, if given together with the password attribute<br>*`password`*<br>Password to use for HTTP basic authentication, if given together with the user attribute<br>*`pattern`*<br>Regular expression according to `java.util.regex.Pattern` to be used for verifying the response<br>*`encoding`*<br>Character encoding for parsing the response, if the pattern attribute is given<br>*`maxSize`* |

| **Http** | |
|---|---|
| | Maximum permitted size of the response in characters (default 65536), if the pattern attribute is given |
| Error codes | 0 (ok)<br><br>10 (unexpected_error)<br><br>11 (timeout)<br><br>12 (error)<br><br>13 (respawning_too_fast)<br><br>21 (io_error) |

*Table 4.16. JMX Action*

| **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.<br><br>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.<br><br>You might also implement a custom converter by extending `com.coremedia.watchdog.impl.CodeConverter`. See the Javadoc for more details. |
| Attribute | *`serviceUrl`*<br><br>A URL to connect to a JMX server<br><br>*`objectName`*<br><br>The qualified name of the MBean.<br><br>*`attributeName`*<br><br>The name of the MBean's attribute |

| **JMX** | |
|---|---|
| | *defaultCode*<br><br>The watchdog code to use if none of the converters is responsible<br><br>*username*<br><br>The user name for the JMX connection (optional).<br><br>*password*<br><br>The password for the JMX connection (optional).<br><br>*Nested elements*<br><br>Each converter needs to be specified as a <Converter> and has to provide its implementation class by "class" attribute. If the converter implementation provides additional configurable attributes ("setter methods"), these attributes might be specified as well. |
| Error codes | The error codes depend on the configured converters or the default code is returned. |

## Example

```
<Jmx name="myJmxAction"
          serviceUrl="service:jmx:jmxmp://localhost:5555/"

objectName="com.coremedia:type=ProactiveEngine,application=caefeeder"

          attributeName="HeartBeat"
          defaultCode="ok">
    <Converter
class="com.coremedia.watchdog.impl.NumberRangeConverter"
                    min="10" max="29999" code="ok"/>
    <Converter
class="com.coremedia.watchdog.impl.NumberRangeConverter"
                    min="30000" code="error"/>
</Jmx>
```

The JMX action "myJmxAction" retrieves the attribute `HeartBeat` of the MBean `com.coremedia:type=ProactiveEngine,application=caefeeder`. If the attribute value is between 10 and 29999 then a watchdog code "ok" is returned. A value greater or equal "30000" results in a code "error". The default code "ok" is used if no converter applies, if the value is lower than 10, for instance.

*Table 4.17. ProcessStatus*

| **ProcessStatus** | |
|---|---|
| Description | This action checks whether a specified number of instances of a process definition with a given name runs in the *Workflow* |

| **ProcessStatus** | |
|---|---|
| | *Server*. It is also checked that these instances do not contain escalated tasks. |
| Attribute | *`url`*<br><br>URL where to get the IOR of the *Workflow Server*<br><br>*`user`*<br><br>CoreMedia user name to log on to the server<br><br>*`domain`*<br><br>Domain of the user<br><br>*`password`*<br><br>CoreMedia user password to log on to the server<br><br>*`processName`*<br><br>The name of the process definition. The predefined workflows have the following process names: GlobalSearchAndReplace, SimplePublication, ThreeStepPublication, TwoStepApproval, TwoStepPublication.<br><br>*`minCount`*<br><br>The minimum number of instances to expect, typically 1<br><br>*`maxCount`*<br><br>The maximum number of instances to expect |
| Error Codes | 0 (ok)<br><br>10 (unexpected_error)<br><br>11 (timeout)<br><br>12 (error)<br><br>13 (respawning_too_fast)<br><br>21 (io_error)<br><br>32 (invalid_login)<br><br>91 (escalated)<br><br>92 (too_few_instances)<br><br>93 (too_many_instances) |

*Table 4.18. Action ServerMode*

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

*Table 4.19. Server Query*

| **ServerQuery** | |
|---|---|
| Description | 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. |

| **ServerQuery** | |
|---|---|
| Attribute | *`url`*<br>URL where to get the IOR of the server<br>*`user`*<br>CoreMedia user name to log on to the server<br>*`domain`*<br>Domain of the user.<br>*`password`*<br>CoreMedia user password to log on to the server |
| Error Codes | 0 (ok)<br>11 (timeout)<br>12 (error)<br>13 (respanwing_too_fast)<br>21 (io_error)<br>31 (no_licenses)<br>32 (invalid_login)<br>33 (corba_error)<br>41 (repository_error)<br>51 (query_malformed)<br>52 (query_failed) |

*Table 4.20. Action ServiceStatus*

| **ServiceStatus** | |
|---|---|
| Description | This action checks a certain service offered by the CoreMedia server, for example the replicator of a *Replication Live Server*. |
| Attribute | *`url`*<br>URL where to get the IOR of the server<br>*`user`*<br>CoreMedia user name to log on to the server<br>*`domain`* |

| **ServiceStatus** | |
|---|---|
| | Domain of the user.<br>*`password`*<br>CoreMedia user password to log on to the server<br>*`service`*<br>Name of the checked service. Possible values are:<br>→ replicator<br>→ adminlogin<br>→ userlogin |
| Error codes | 0 (ok)<br>10 (unexpected_error)<br>11 (timeout)<br>12 (error)<br>13 (respawning_too_fast)<br>21 (io_error)<br>31 (no_licenses)<br>32 (invalid_login)<br>33 (corba_error)<br>81 (service_stopped)<br>82 (service_failed)<br>83 (service_disabled) |

*Table 4.21. Action Script*

| **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.<br>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 |

| **Script** | |
|---|---|
| | 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 | *`command`*<br><br>Name of the executable command |
| Error codes | 0 (ok) |

*Table 4.22. WorkflowServerQuery*

| **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 | *`url`*<br><br>URL where to get the IOR of the server<br><br>*`user`*<br><br>CoreMedia user name to log on to the server<br><br>*`domain`*<br><br>Domain of the user.<br><br>*`password`*<br><br>CoreMedia user password to log on to the server |
| Error Codes | 0 (ok)<br><br>10 (unexpected_error)<br><br>11 (timeout)<br><br>12 (error)<br><br>13 (respanwing_too_fast)<br><br>21 (io_error)<br><br>32 (invalid_login)<br><br>41 (repository_error)<br><br>52 (query_failed) |

## 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:

*Table 4.23. Attributes of the Edge element*

| Attribute | Optional | Description |
|---|---|---|
| `from` | | The name of the action from which the edge starts. |
| `to` | | The name of the action which follows. |
| `code` | | Result code (numerical entry or symbolic name). If this value equals the result code of the action named in the `from` attribute, then the action named in the `to` attribute is executed. Edges may not start from the same action with the same error code to different target actions. To avoid defining an `Edge` 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"`. |
| `delay` | x | This attribute specifies the delay in seconds until the subsequent action is executed. The default value is 10 seconds. |

**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"/ >
```

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://local
host:44441/coremedia/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:44441/coremedia/ior** and logs in with user name "watchdog" and password "watchdog".

```
<ServerQuery    name="WS-CorbaReQuery"    url="http://local
host:44441/coremedia/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 cm7-cms-tomcat
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_li
censes" 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_li
censes" 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="respawn
ing_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>
```

## 4.8.4 Watchdog Result Codes

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

*Table 4.24. watchdog result codes*

| Result code | Symbolic name | Description |
|---|---|---|
| 0 | "ok" | The application works properly without errors. |
| 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 | "respawn-ing_too_fast" | An action was executed to often in a specified time interval (see Section “Action Elements” [54]). |
| 21 | "io_error" | An action has failed with an I/O error. |
| 31 | "no_licenses" | An action has failed because there was no free li-cense. |
| 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. |

| Result code | Symbolic name | Description |
|---|---|---|
| 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. |
| 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 *Replication Live Server*. |
| 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_toomanyinstaces" | The `ProcessStatus` action has found too many instances of the observed process definition. |

## 4.9 JMX Management

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

All resources are registered using Spring's ability to register and export MBeans to an MBean server. 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 infrastructure such as a remote connector server.

The remote connector server can be configured by providing the property `management.server.remote.url` either in `WEB-INF/application.properties`, as a system property, or as a JNDI property `java:comp/env/coremedia/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 CoreMedia Digital Experience Platform 8 Developer 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 *CAE Feeder* makes content beans searchable by sending their data to the *Search Engine*, which adds it to the index. |
| Content Application Engine (CAE) | The *Content Application Engine* (*CAE*) is a framework for developing content applications with *CoreMedia CMS*.<br><br>While it focuses on web applications, the core frameworks remain usable in other environments such as standalone clients, portal containers or web service implementations.<br><br>The CAE uses the Spring Framework for application setup and web request processing. |
| Content Bean | A content bean defines a business oriented access layer to the content, that is managed in *CoreMedia CMS* and third-party systems. Technically, a content bean is a Java object that encapsulates access to any content, either to CoreMedia CMS content items or to any other kind of third-party systems. Various CoreMedia components like the CAE Feeder or the data view cache are built on this layer. For these components the content beans act as a facade that hides the underlying technology. |
| Content Delivery Environment | The *Content Delivery Environment* is the environment in which the content is delivered to the end-user.<br><br>It may contain any of the following modules:<br><br>→ *CoreMedia Master Live Server*<br>→ *CoreMedia Replication Live Server*<br>→ *CoreMedia Content Application Engine*<br>→ *CoreMedia Search Engine*<br>→ *Elastic Social* |

|  |  |
|---|---|
|  | → *CoreMedia Adaptive Personalization* |
| Content Feeder | The *Content Feeder* is a separate web application that feeds content items of the CoreMedia repository into the *CoreMedia Search Engine*. Editors can use the *Search Engine* to make a full text search for these fed items. |
| Content item | In *CoreMedia CMS*, content is stored as self-defined content items. Content items are specified by their properties or fields. Typical content properties are, for example, title, author, image and text content. |
| Content Management Environment | The *Content Management Environment* is the environment for editors. The content is not visible to the end user. It may consist of the following modules:<br>→ *CoreMedia Content Management Server*<br>→ *CoreMedia Workflow Server*<br>→ *CoreMedia Importer*<br>→ *CoreMedia Site Manager*<br>→ *CoreMedia Studio*<br>→ *CoreMedia Search Engine*<br>→ *CoreMedia Adaptive Personalization*<br>→ *CoreMedia CMS for SAP Netweaver ® Portal*<br>→ *CoreMedia Preview CAE* |
| Content Management Server | Server on which the content is edited. Edited content is published to the Master Live Server. |
| Content Repository | *CoreMedia CMS* manages content in the Content Repository. Using the Content Server or the UAPI you can access this content. Physically, the content is stored in a relational database. |
| Content Server | *Content Server* is the umbrella term for all servers that directly access the CoreMedia repository:<br>*Content Servers* are web applications running in a servlet container.<br>→ *Content Management Server*<br>→ *Master Live Server*<br>→ *Replication Live Server* |

| | |
|---|---|
| Content type | A content type describes the properties of a certain type of content. Such properties are for example title, text content, author, ... |
| Contributions | Contributions are tools or extensions that can be used to improve the work with *CoreMedia CMS*. They are written by CoreMedia developers - be it clients, partners or CoreMedia employees. CoreMedia contributions are hosted on Github at https://github.com/coremedia-contributions. |
| Controm Room | *Controm Room* is a *Studio* plugin, which enables users to manage projects, work with workflows, and collaborate by sharing content with other *Studio* users. |
| CORBA (Common Object Request Broker Architecture) | The term *CORBA* refers to a language- and platform-independent distributed object standard which enables interoperation between heterogenous applications over a network. It was created and is currently controlled by the Object Management Group (OMG), a standards consortium for distributed object-oriented systems.<br><br>CORBA programs communicate using the standard IIOP protocol. |
| CoreMedia Studio | *CoreMedia Studio* is the working environment for business specialists. Its functionality covers all of the stages in a web-based editing process, from content creation and management to preview, test and publication.<br><br>As a modern web application, *CoreMedia Studio* is based on the latest standards like Ajax and is therefore as easy to use as a normal desktop application. |
| Dead Link | A link, whose target does not exists. |
| DTD | A Document Type Definition is a formal context-free grammar for describing the structure of XML entities.<br><br>The particular DTD of a given Entity can be deduced by looking at the document prolog:<br><br>`<!DOCTYPE coremedia SYSTEM "http://www.coremedia.com/dtd/coremedia.dtd"`<br><br>There're two ways to indicate the DTD: Either by Public or by System Identifier. The System Identifier is just that: a URL to the DTD. The Public Identifier is an SGML Legacy Concept. |
| Elastic Social | *CoreMedia Elastic Social* is a component of *CoreMedia CMS* that lets users engage with your website. It supports features like comments, rating, likings on your website. *Elastic Social* is integrated into *CoreMedia Studio* so editors can moderate user generated content from their common workplace. *Elastic Social* bases on NoSQL technology and offers nearly unlimited scalability. |

| | |
|---|---|
| EXML | EXML is an XML dialect supporting the declarative development of complex Ext JS components. EXML is Jangaroo's equivalent to Adobe Flex MXML and compiles down to Actions Script. |
| 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. |
| Home Page | The main entry point for all visitors of a site. Technically it is often referred to as root document and also serves as provider of the default layout for all subpages. |
| IETF BCP 47 | Document series of *Best current practice* (BCP) defined by the Internet Engineering Task Force (IETF). It includes the definition of IETF language tags, which are an abbreviated language code such as en for English, pt-BR for Brazilian Portuguese, or nan-Hant-TW for Min Nan Chinese as spoken in Taiwan using traditional Han characters. |
| Importer | Component of the CoreMedia system for importing external content of varying format. |
| IOR (Interoperable Object Reference) | A CORBA term, *Interoperable Object Reference* refers to the name with which a CORBA object can be referenced. |
| Jangaroo | *Jangaroo* is a JavaScript framework developed by CoreMedia that supports 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.<br><br>It consists of HTML code fragments in which Java code can be embedded. |
| Locale | Locale is a combination of country and language. Thus, it refers to translation as well as to localization. Locales used in translation processes are typically represented as IETF BCP 47 language tags. |
| Master Live Server | The *Master Live Server* is the heart of the *Content Delivery Environment*. It receives the published content from the *Content Management Server* and makes it available to the *CAE*. If you are using the *CoreMedia Multi-Site Management Extension* you may use multiple *Master Live Server* in a CoreMedia system. |

| | |
|---|---|
| Master Site | A master site is a site other localized sites are derived from. A localized site might itself take the role of a master site for other derived sites. |
| MIME | With Multipurpose Internet Mail Extensions (MIME), the format of multi-part, multimedia emails and of web documents is standardised. |
| 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. |
| Property | In relation to CoreMedia, properties have two different meanings:<br><br>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 items depends on the content type.<br><br>In connection with the configuration of CoreMedia components, the system behavior of a component is determined by properties. |
| Replication Live Server | The aim of the *Replication Live Server* is to distribute load on different servers and to improve the robustness of the *Content Delivery Environment*. The *Replication Live Server* is a complete Content Server installation. Its content is an replicated image of the content of a *Master Live Server*. The *Replication Live Server* updates its database due to change events from the *Master Live Server*. You can connect an arbitrary number of *Replication Live Servers* to the *Master Live Server*. |
| Resource | A folder or a content item in the CoreMedia system. |
| ResourceURI | A ResourceUri uniquely identifies a page which has been or will be created by the *Active Delivery Server*. The ResourceUri consists of five components: Resource ID, Template ID, Version number, Property names and a number of key/value pairs as additional parameters. |
| Responsive Design | Responsive design is an approach to design a website that provides an optimal viewing experience on different devices, such as PC, tablet, mobile phone. |
| Site | A site is a cohesive collection of web pages in a single locale, sometimes referred to as localized site. In *CoreMedia CMS* a site especially consists of a site folder, a site indicator and a home page for a site.<br><br>A typical site also has a master site it is derived from. |

| | |
|---|---|
| Site Folder | All contents of a site are bundled in one dedicated folder. The most prominent document in a site folder is the site indicator, which describes details of a site. |
| Site Indicator | A site indicator is the central configuration object for a site. It is an instance of a special content type, most likely `CMSite`. |
| Site Manager | Swing component of CoreMedia for editing content items, managing users and workflows. |
| Site Manager Group | Members of a site manager group are typically responsible for one localized site. Responsible means that they take care of the contents of that site and that they accept translation tasks for that site. |
| Template | In CoreMedia, JSPs used for displaying content are known as Templates.<br><br>OR<br><br>In *Blueprint* a template is a predeveloped content structure for pages. Defined by typically an administrative user a content editor can use this template to quickly create a complete new page including, for example, navigation, predefined layout and even predefined content. |
| Translation Manager Role | Editors in the translation manager role are in charge of triggering translation workflows for sites. |
| User Changes web application | The *User Changes* web application is a *Content Repository* listener, which collects all content, modified by *Studio* users. This content can then be managed in the *Control Room*, as a part of projects and workflows. |
| Version history | A newly created content item receives the version number 1. New versions are created when the content item is checked in; these are numbered in chronological order. |
| Weak Links | In general *CoreMedia CMS* always guarantees link consistency. But links can be declared with the *weak* attribute, so that they are not checked during publication or withdrawal.<br><br>Caution! Weak links may cause dead links in the live environment. |
| WebDAV | WebDAV stands for World Wide Web Distributed Authoring and Versioning Protocol. It is an extension of the Hypertext Transfer Protocol (HTTP), which offers a standardised method for the distributed work on different data via the internet. This adds the possibility to the CoreMedia system to easily access CoreMedia resources via external programs. A WebDAV enabled application like Microsoft Word is thus able to open Word documents stored in the CoreMedia system. For further information, see http://www.webdav.org. |

| | |
|---|---|
| Workflow | A workflow is the defined series of tasks within an organization to produce a final outcome. Sophisticated applications allow you to define different workflows for different types of jobs. So, for example, in a publishing setting, a document might be automatically routed from writer to editor to proofreader to production. At each stage in the workflow, one individual or group is responsible for a specific task. Once the task is complete, the workflow software ensures that the individuals responsible for the next task are notified and receive the data they need to execute their stage of the process. |
| Workflow Server | The *CoreMedia Workflow Server* is part of the Content Management Environment. It comes with predefined workflows for publication and global-search-and-replace but also executes freely definable workflows. |
| XLIFF | XLIFF is an XML-based format, standardized by OASIS for the exchange of localizable data. An XLIFF file contains not only the text to be translated but also metadata about the text. For example, the source and target language. *CoreMedia Studio* allows you to export content items in the XLIFF format and to import the files again after translation. |

# Index