CoreMedia Content Cloud v13 Upgrade Guide

CoreMedia Content Cloud – v13

CoreMedia GmbH

Altes Klöpperhaus, 5. OG

Rödingsmarkt 9

20459 Hamburg

March 11, 2026 (Release 2512.0)

List of Figures

List of Tables

1. Preface

This guide describes the upgrade from a *CoreMedia Content Cloud (CMCC) v12 Blueprint* system to a *CMCC v13 Blueprint* system.

This guide is intended for readers involved in planning and executing an upgrade.

1.1 Using This Guide

The Upgrade Guide is structured so that you can focus on the information relevant to your role and system.

- Chapter 2, *CoreMedia Content Cloud v13* [13] High-level overview of the main changes and benefits in CoreMedia Content Cloud v13.
- Chapter 3, *Upgrade Information Overview* [15] General information relevant to all upgrades. Recommended reading for everyone involved.
- Chapter 4, *Prerequisites for the Upgrade* [18] Required development environment and software for building and upgrading the workspace.
- Chapter 5, *Changes in Supported Environments* [20] Changes in supported environments, such as databases and platforms.
- Chapter 6, *Added and Removed Components and Features* [24] Removed components and features. Review carefully for breaking changes.
- Chapter 7, *Detailed Overview of Changes* [29] Detailed information on specific changes, including new, deprecated, and removed components, third-party software updates, and new operation and configuration concepts. Focus on the topics relevant to your system and upgrade path. Each topic states: • who is affected, • whether action is required or optional, and • how to adapt your system.

1.2 Audience

This manual is intended for developers, architects, and project managers who plan to upgrade from *CoreMedia Content Cloud* v12 to *CoreMedia Content Cloud* v13.

1.3 Typographic Conventions

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

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

Table 1.1. Typographic conventions

In addition, these symbols can mark single paragraphs:

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

Table 1.2. Pictographs

1.4 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 1.4.1, “Registration” [6] for details on how to register.

NOTE

CoreMedia User Orientation for CoreMedia Developers and Partners

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

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

- Section 1.4.1, “Registration” [6] describes how to register for the usage of the services.
- Section 1.4.2, “CoreMedia Releases” [7] describes where to find the download of the software.
- Section 1.4.3, “Documentation” [8] describes the CoreMedia documentation. This includes an overview of the manuals and the URL where to find the documentation.
- Section 1.4.4, “CoreMedia Training” [11] describes CoreMedia training. This includes the training calendar,the curriculum and certification information.
- Section 1.4.5, “CoreMedia Support” [11] describes the CoreMedia support.

1.4.1 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 1.4.5, “CoreMedia Support” [11]) by email to request further access depending on your customer, partner or freelancer status so that you can use the CoreMedia services.

1.4.2 CoreMedia Releases

Downloading and Upgrading the Blueprint Workspace

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

https://releases.coremedia.com/cmcc-13

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

NOTE

If you encounter a 404 error then you are probably not logged in at GitHub or do not have sufficient permissions yet. See Section 1.4.1, "Registration" [6] for details about the registration process. If the problems persist, try clearing your browser cache and cookies.

Maven artifacts

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

https://repository.coremedia.com

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

npm packages

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

https://repository.coremedia.com/nexus/repository/coremedia-npm/

The `.npmrc` file must be configured to be able to use the registry (see Section 3.1, "Prerequisites" in *Blueprint Developer Manual*).

License files

You need license files to run the CoreMedia system. Contact the support (see Section 1.4.5, “CoreMedia Support” [11]) to get your licences.

1.4.3 Documentation

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

https://documentation.coremedia.com

The manuals have the following content and use cases:

Manual	Audience	Content
Blueprint Developer Manual	Developers, architects, administrators	This manual gives an overview over the structure and features of *CoreMedia Content Cloud*. It describes the content type model, the *Studio* extensions, folder and user rights concept and many more details. It also describes administrative tasks for the features. It also describes the concepts and usage of the project workspace in which you develop your CoreMedia extensions. You will find a description of the Maven structure, the virtualization concept, learn how to perform a release and many more.
Connector Manuals	Developers, administrators	This manuals gives an overview over the use cases of the eCommerce integration. It describes the deployment of the Commerce Connector and how to connect it with the CoreMedia and eCommerce system.
Content Application Developer Manual	Developers, architects	This manual describes concepts and development of the *Content Application Engine (CAE)*. You will learn how to write 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

Manual	Audience	Content
		type model which lies at the heart of a CoreMedia system, about user and rights management, database configuration, and more.
Deployment Manual	Developers, architects, administrators	This manual describes the concepts and usage of the CoreMedia deployment artifacts. That is the deployment archive and the Docker setup. You will also find an overview of the properties required to configure the deployed system.
Elastic Social Manual	Developers, architects, administrators	This manual describes the concepts and administration of the *Elastic Social* module and how you can integrate it into your websites.
Frontend Developer Manual	Frontend Developers	This manual describes the concepts and usage of the Frontend Workspace. You will learn about the structure of this workspace, the CoreMedia themes and bricks concept, the CoreMedia Freemarker facade API, how to develop your own themes and how to upload your themes to the CoreMedia system.
Headless Server Developer Manual	Frontend Developers, administrators	This manual describes the concepts and usage of the *Headless Server*. You will learn how to deploy the Headless Server and how to use its endpoints for your sites.
Multi-Site Manual	Developers, Multi-Site Administrators, Editors	This manual describes different otions to desgin your site hierarchy with several languages. It also gives guidance to avoid common pitfalls during your work with the multi-site feature.
Operations Basics Manual	Developers, administrators	This manual describes some overall concepts such as the communication between the components, how to set up secure connections, how to start application.
Search Manual	Developers, architects, administrators	This manual describes the configuration and customization of the *CoreMedia Search Engine* and the two feeder applications: the *Content Feeder* and the *CAE Feeder*.

Manual	Audience	Content
Studio Developer Manual	Developers, architects	This manual describes the concepts and extension of *CoreMedia Studio*. You will learn about the underlying concepts, how to use the development environment and how to customize *Studio* to your needs.
Studio User Manual	Editors	This manual describes the usage of *CoreMedia Studio* for editorial and administrative work. It also describes the usage of the *Native Personalization* and *Elastic Social* GUI that are integrated into *Studio*.
Studio Benutzerhandbuch	Editors	The Studio User Manual but in German.
Supported Environments	Developers, architects, administrators	This document lists the third-party environments with which you can use the CoreMedia system, Java versions or operation systems for example.
Unified API Developer Manual	Developers, architects	This manual describes the concepts and usage of the *CoreMedia Unified API*, which is the recommended API for most applications. This includes access to the content repository, the workflow repository and the user repository.
Utilized Open Source Software & 3rd Party Licenses	Developers, architects, administrators	This manual lists the third-party software used by CoreMedia and lists, when required, the licence texts.
Workflow Manual	Developers, architects, administrators	This manual describes the *Workflow Server*. This includes the administration of the server, the development of workflows using the XML language and the development of extensions.

Table 1.3. CoreMedia manuals

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

Email: documentation@coremedia.com

1.4.4 CoreMedia Training

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

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

https://www.coremedia.com/training

Contact the training department at the following email address:

Email: training@coremedia.com

1.4.5 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:

https://support.coremedia.com/

Do not forget to request further access via email after your initial registration as described in Section 1.4.1, “Registration” [6]. The support email address is:

Email: support@coremedia.com

Create a support request

Support request

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

- Which CoreMedia component(s) did the problem occur with (include the release number)?
- Which database is in use (version, drivers)?
- Which operating system(s) is/are in use?
- Which Java environment is in use?

- Which customizations have been implemented?
- A full description of the problem (as detailed as possible)
- Can the error be reproduced? If yes, give a description please.
- How are the security settings (firewall)?

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

Support checklist

To put it in a nutshell, CoreMedia needs:

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

Log files

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

Which Log File?

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

Where do I Find the Log Files?

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

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

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

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

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

2. CoreMedia Content Cloud v13

This chapter provides a high-level overview of ***CoreMedia Content Cloud* v13** and highlights the most important changes compared to v12.

It is intended to help you understand the **overall scope and direction** of the release before diving into the detailed upgrade information in later chapters. Use this chapter to get the big picture and to identify which areas are most relevant for your upgrade.

You do not need to read this guide from beginning to end. All chapters can be read independently. Use the structure of this guide and the search function to quickly navigate to topics that matter for your individual upgrade.

New Features - The Highlights

CoreMedia Content Cloud v13 introduces improvements across several areas, with a strong focus on usability and efficiency. A major emphasis of this release is the **Studio user interface**, with enhancements aimed at better supporting users in their daily work and making content creation and management more efficient.

Key highlights include:

- **Semantic Search**

  CoreMedia Studio has been extended with semantic search capabilities. This allows you to find content items based on their meaning and context rather than just matching keywords. This new feature contains the usage part in Studio and the configuration part. It requires an account at an embedding provider which may incur addition costs.

Technical Changes - Overview

In addition to user-facing improvements, CoreMedia Content Cloud v13 includes technical changes that may affect development, operations, and integrations.

The following list shows the most important changes in terms of upgrade effort. Chapter 7, *Detailed Overview of Changes* [29] lists all changes in detail. See the https://releases.coremedia.com/cmcc-13/artifacts/CMCC 13 - Supported Environments.pdf document for the list of supported databases, operating systems, and browsers of this release.

- **Replacing CORBA with gRPC**

  The migration from CORBA to gRPC modernizes the inter-service communication layer. This change requires updates to network configurations and may affect custom integrations that rely on CORBA. See Section 7.3, "Adding gRPC Support for Component Communication" [36] for details.

- **Upgrade to Spring Boot 4**

  Spring Boot 4 is required to build production-ready applications using Spring Framework 7. The upgrade was necessary because of the shortened Support Lifecycle of Spring Boot.

- **Upgrade to Java 21**

  Upgrading to Spring Boot 3.2 and Spring Framework 6.1 required upgrading to Java 21 as the minimum Java version as described in the official Spring Framework documentation.

- **Removal of XML Importer**

  The legacy XML Importer has been removed. If your workflow depends on this component, you must migrate to alternative content import mechanisms.

Refer to **Chapter 7, Detailed Overview of Changes**, for in-depth information and concrete migration steps.

3. Upgrade Information Overview

The upgrade information for CoreMedia Content Cloud is divided into:

- *General upgrade information*, which applies to all upgrades, and
- *Specific upgrade information*, which applies to upgrading from CMCC v12 to CMCC v13.

CoreMedia strongly recommends that you review the general upgrade information before starting the upgrade process.

3.1 General Upgrade Information

General information about upgrading CoreMedia Content Cloud is available on the documentation site. These resources describe upgrade concepts and best practices that are independent of a specific target version.

The following topics are particularly relevant:

CoreMedia Release Cycle and LTS Strategy	Describes the CoreMedia release cycle and long-term support strategy.
Why You Should Perform an Upgrade	Explains technical and business reasons for upgrading your workspace.
How to Manage Upgrade Tasks	Provides guidance on planning, structuring, and tracking upgrade activities.
General Upgrading Tasks	Describes common upgrade steps, such as upgrading to the latest patch level of your current version, removing outdated components, and preparing your workspace for the new Blueprint.
Upgrading with Git	Explains how to merge a new version of the CoreMedia Blueprint into an existing workspace using Git.
Upgrade Guides	Provides access to all available CoreMedia upgrade guides, including this guide.

3.2 Specific Upgrade Information (v12 → v13)

This guide contains detailed information specific to upgrading from CoreMedia Content Cloud v12 to v13. It focuses on changes that may require action in your workspace, environment, or customizations.

Chapter 4, *Prerequisites for the Upgrade* [18]	Describes the software and tools required to build and upgrade the workspace.
Chapter 5, *Changes in Supported Environments* [20]	Lists changes in supported environments, such as databases or platforms, that may affect your upgrade planning.
Chapter 6, *Added and Removed Components and Features* [24]	Lists components, integrations, and features that were removed in v13. Review this chapter early to identify breaking changes and required cleanup steps.
Chapter 7, *Detailed Overview of Changes* [29]	Describes all relevant changes in detail, organized by component. This includes information about required or optional actions, changes to configuration and deployment, and impacts on customizations.

4. Prerequisites for the Upgrade

Before upgrading to **CoreMedia Content Cloud v13**, ensure that your system and development environment meets the following prerequisites. CoreMedia strongly recommends starting the upgrade from the latest version of *CoreMedia Content Cloud* v12. This chapter lists the required software and access prerequisites needed to build and upgrade the workspace.

The following tools must be installed before starting the upgrade:

- **Git >= 2.25.0**

  The guide uses the Git command **restore** which is only available since Git version 2.25.0.
- **NodeJS 24.x**

  In order to compile the Studio Client workspace and all frontend related components, you need to install NodeJS in the latest version 20.x.
- **pnpm 10.16.0**

  In order to compile the Studio Client workspace and all frontend related components, you need to install pnpm as the package manager. Simply call `npm install -g pnpm@10.16.0` on the commandline.

Configuring Access to CoreMedia npm Repository

CoreMedia Content Cloud v13 uses a CoreMedia-provided npm registry for distributing CoreMedia npm packages. To access this registry, you must configure authentication using a Nexus access token.

Prerequisites:

Access to the CoreMedia Nexus at https://repository.coremedia.com

1. Creating an Access Token

   Log into Nexus at https://repository.coremedia.com with your credentials
2. Click **User Token** in the user menu at the top right.

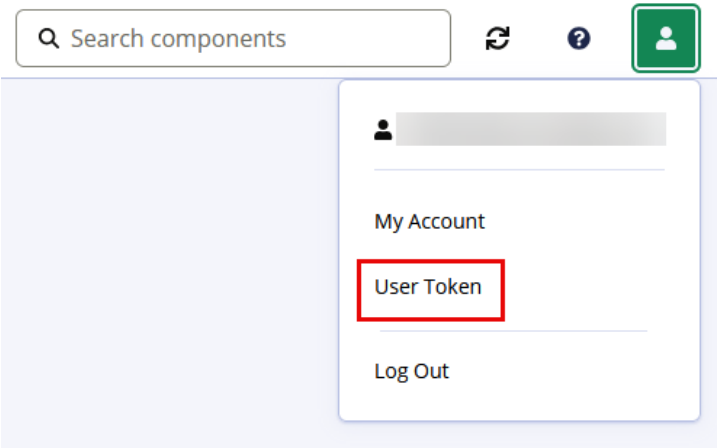

Figure 4.1. User token menu

3. Click **Access User Token** and re-enter your password in the dialog that opens up.
4. Copy the last entry **Use the following for a base64 representation of "user:password"** to the clipboard.

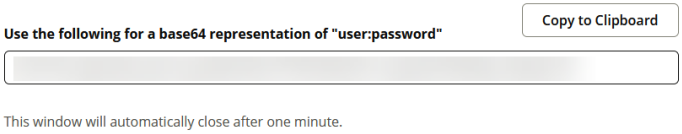

Figure 4.2. Copy user token

5. Use the following pnmp commands to set up the registry and authentication in the `.npmrc` file. Replace the base64 encoded string in the last command, with the one you copied from Nexus :

```
pnpm config set "@coremedia:registry"
"https://repository.coremedia.com/nexus/repository/coremedia-npm/"
pnpm config set "@jangaroo:registry"
"https://repository.coremedia.com/nexus/repository/coremedia-npm/"
pnpm config set
"//repository.coremedia.com/nexus/repository/coremedia-npm/:_auth"
hjFtZUBjb5K4bWVkaWEuY29tOmFkbWluNJKz
```

5. Changes in Supported Environments

This chapter lists third-party software that is no longer supported by *CoreMedia Content Cloud* or which has been added.

See the Supported Environments at https://releases.coremedia.com/cmcc-13/artifacts/CMCC 13 - Supported Environments.pdf for details.

5.1 Added Support

5.1.1 Oracle 23ai

CMCC 13 now supports Oracle 23ai.

5.1.2 Java 21

CoreMedia 13 requires Java 21. Java 21 is the next Long Term Support version of the Java platform following after Java 17. The switch to Java 21 has the following implications:

Maven Plugins have been updated

This includes especially a required update of the maven-dependency-plugin. It should be noted that the updated version provides stricter dependency checks, which may mean that you need to adapt your dependencies in Maven POMs to comply with the new requirements. The updated plugins ship with an update of `com.coremedia:cms-blueprint-parent` to version 3.0.0.

Java Application Base Image updated

A new Java Application Base Image is used for the CMS applications: `coremedia/java-application-base:3.1-cm-21.0-corretto-jre`.

Removed Deprecated Java API Usages

Deprecated API usages have been removed, which includes, but is not limited to:

- `Locale` constructor: Changed to using `Locale.of()`.
- `URL` constructor: Replaced by alternatives like `new URI(...).toURL()`.
- Widened `@cm.template.api annotations`: To address issues in the generated JavaDoc for the template API, fixed missing @cm.template.api annotations.

5.1.3 NodeJS 24

NodeJS 24 (LTS) is now required to build the Frontend and Studio Client workspaces.

5.1.4 pnpm 10.16.0

Along with NodeJS 24, pnpm 10.16.0 is now required to build the Frontend and Studio Client workspaces.

5.1.5 Maven 3.9.11

Supported and required Maven version has been updated to 3.9.11. Also, the Takari Maven Extensions have been removed (`.mvn/extensions.xml`) which means the option `--builder smart` is not available anymore by default.

See https://maven.apache.org/docs/3.9.11/release-notes.html.

5.1.6 Container Operating Systems

Ubuntu with OpenJDK 21 and Amazon Linux with Corretto 21 have been added as new container operating systems.

5.2 Removed Support

5.2.1 Java 17

Java 17 is no longer supported. CoreMedia 13 requires Java 21. See Section 7.4.2.10, “Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl” [?] for details.

5.2.2 Microsoft SQL Server

Support for Microsoft SQL Server (MSSQL) has been completely removed in CMCC 13.

5.2.3 MySQL 8.0

Support for MySQL 8.0 in CMCC 13 has been removed. Support for this version from Oracle will be ended during the lifetime of CMCC 13. CMCC 13 now requires MySQL version 8.4.

5.2.4 PostgreSQL 13

Support for PostgreSQL 13 in CMCC 13 has been removed because PostgreSQL will run out of support during the lifetime of CMCC 13. CMCC 13 now requires PostgreSQL version 16.

5.2.5 MongoDB 6.0 and 7.0 for Elastic Social

Support for MongoDB 6.0 and 7.0 for Elastic Social has been removed. MongoDB for Elastic Social now requires version 8.0.

Support for MongoDB in collaborative components and workflows has been completely removed. You can use an SQL database now.

6. Added and Removed Components and Features

This section lists components and features which have been added or removed in *CoreMedia Content Cloud*. In most cases, CoreMedia recommends that you remove the components from your workspace before upgrading to the new version.

6.1 Added Components and Features

6.1.1 gRPC for Internal Component Communication

For internal communication between CoreMedia CMCC components, gRPC has been added as an alternative to CORBA. Communication via CORBA is still available and the default. As long as you plan to stick with CORBA, no changes to your code or operations procedures are required. Should you want to switch to gRPC as your communications protocol between CMCC components, see 7. 16. gRPC for Internal Communication .

6.1.2 Additional Security for Content Server HTTP Endpoints

Each Content Server provides HTTP endpoints for blob up- and download as well as for processor usage data. These are internally used endpoints that are not supported for use by custom clients. You may configure whether authentication data for those endpoints is to be sent as part of the URL (which was the only way prior to CMCC 13) or as a request header field. While sending the token as a query parameter is inherently insecure, it is the default to keep backward compatibility. Unless connection to an older server (prior to CMCC 13) is required, it is recommended to set the corresponding properties to "false".

See Deployment Manual, section 4.4.7 Securing Session Authentication for Content Server HTTP Endpoints for details of configuration.

6.1.3 Session Check for Blob Download from Content Server Can Be Disabled

Blob Downloads from Content Servers are verified for a valid login session on each call. With a new configuration option, this session validation can be disabled.

See Section 6.1.3, “Session Check for Blob Download from Content Server Can Be Disabled” [57] for details.

6.1.4 Semantic Search

CoreMedia Studio has been extended with semantic search capabilities. This allows you to find content items based on their meaning and context rather than just matching keywords. This new feature contains the usage part in Studio and the configuration part. It requires an account at an embedding provider which may incur addition costs.

- Usage: See section "Semantic Search" in *Studio User Manual* for details.
- Configuration: See Section 4.1.4, "Semantic Search" in *Search Manual* for a general description and the configuration of the Embedding Service and Section 4.4, "Configure Search for Studio" in *Search Manual* for a general description of the configuration and Section 3.4, "Studio Properties" in *Deployment Manual* for a description of the required configuration properties (search for "semantic").

6.1.5 Backend Support for Variants

In order to support personalization variants, a new meta-property `variant` was added to Content items. It is immutable once set during content creation. It is accessible through `isVariant()` and its inverse `isBaseline()`. For a baseline content item, multiple variants may be created. Variants are created with the same content type as their baseline. They are used for delivering A/B tests or specialized content for user segments.

6.2 Removed Components and Features

6.2.1 Removed XML Importer

Support for the XML Importer was already deprecated in CMCC 12 and has now been removed. The XML Importer allowed importing content from XML files into CoreMedia Studio. CoreMedia recommends to use the Ingest Service or custom UAPI clients instead.

6.2.2 Removed Build Profile turbojpeg

Profile turbojpeg has been removed from the Blueprint workspace. It was experimental and unsupported for several years. Built-in image transformation in CAE and Headless Server continues to work as before. If in need for high-performance image transformation, consider using CoreMedia's Image Transformation Service.

6.2.3 Removed MongoDB support for Collaborative Features

The CoreMedia collaborative features Control Room plugin, Notifications plugin, User Changes application and extensions of the Workflow Server used MongoDB to store their data. CoreMedia replaced MongoDB for this use cases with an SQL persistence layer. So you can use, for example, the same database as for your CoreMedia repository. This simplifies your setup.

See Section 4.5.3, "Migration to SQL Persistence" in *Operations Basics* for migration steps.

6.2.4 Removal of Analytics Connectors

The integration of Analytics Connectors based on Elastic Core has been removed, that is, extension `es-alx` and parts of extension `alx`, including the analytics content types. Functionality for website tracking is still available and the corresponding documentation has moved to the CoreMedia Blueprint manual.

See Section 7.4.2.10, “Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl” [?] for details on the removed content types and features.

6.2.5 Workflow App Based on ExtJs Replaced By React Workflow App

The Workflow App has been replaced 1:1 by a new React implementation instead of ExtJs. It goes along with some visual improvements, otherwise the UI and UX is basically the same. For a detailed listing of changed, see Section 7.4.1, “ExtJs Workflow App Replaced by React Workflow App” [39].

7. Detailed Overview of Changes

This chapter provides an overview of the improvements in *CoreMedia Content Cloud* v13 over *CoreMedia Content Cloud* v12.

- Section 7.1, "Updated Third-Party Libraries" [30] lists updated third-party libraries.
- Section 7.2, "API Changes" [33] lists all API changes in tables and describes some other API changes of *CoreMedia Content Cloud* v12 in separate sections.
- Section 7.4, "Studio Client Changes" [39] describes changes in the Studio Client.
- Section 7.4.2.10, "Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl" [?] describes changes in the Content Servers.
- Section 7.4.2.10, "Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl" [?] describes changes in the Content Application Engine.
- Section 7.4.2.10, "Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl" [?] describes changes in the *CoreMedia Content Cloud* v12 Headless Server.
- Section 7.4.2.10, "Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl" [?] describes changes in the different CoreMedia commerce integrations.
- Section 7.4.2.10, "Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl" [?] describes changes in the *CoreMedia Blueprint*.
- Section 7.4.2.10, "Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl" [?] describes changes in the Frontend workspace. This also includes frontend related topics in the *CoreMedia Blueprint*.

A complete list of changes is available on the https://documentation.coremedia.com/cmcc-13/release-notes pages.

7.1 Updated Third-Party Libraries

This section lists updated third-party libraries in *CoreMedia Content Cloud* v13. You can find more API changes in the other change chapters.

7.1.1 Spring Boot 4 / Springframework 7

The Spring dependencies were upgraded to the Spring Boot 4 / Spring 7 baseline. With these upgrades, some further dependencies had to be upgraded as well.

The following Spring dependencies were upgraded:

- Spring Boot -> 4.0.0, see also Spring Boot 4.0 Release Notes and Spring Boot 4.0 Migration Guide.
- Spring Framework -> 7.0.1, see also Spring Framework 7.0 Release Notes.
- Spring Security -> 7.0.0, see also What's New in Spring Security 7.0.
- Spring Data BOM -> 2025.1.1, see also Spring Data 2025.1 Release Notes.
- Spring GraphQL -> 2.0.1, see also Spring for GraphQL 2.0.
- Spring Webflow -> 4.0.0.
- Micrometer -> 1.16.1.
- Reactor Project BOM -> 2025.0.1.

Numerous third-party dependencies have also been updated, some of the more noteworthy of which are the following:

- GraphQL -> 25.0
- HIbernate ORM -> 7.1.11.Final
- Hibernate Validator -> 9.1.0.Final
- Jakarta Servlet API -> 6.1.0
- Tomcat -> 11.0.15
- JUnit Jupiter -> 6.0.1

Most notable changes

Module Dependencies

Spring Boot 4.0 has a new modular design and now ships smaller focused modules rather than several large JARs. With this, many Spring Boot dependencies and Java packages have changed and respective references must be applied.

Take extra care of Spring Boot's auto configurations. Many of them have now vanished from the classpaths, because they were moved to module JARs that are not configured as project (compile) dependencies. Some of them weren't actually used, and they don't pollute the classpaths anymore. Others may be required for your application's functionality but now could have vanished. The new module JARs containing these auto configurations may have to be added as additional runtime or test dependencies now. To identify potentially missing auto configurations, you may start your Spring Boot applications or tests before and after upgrade with `-Ddebug=true` and compare the CONDITIONS EVALUATION REPORT in the logs.

Some unwanted auto configurations from the former Spring Boot JARs had to be excluded explicitly. Now their respective module dependencies have been removed so that those exclusions are not needed anymore:

- The MongoAutoConfiguration had to be excluded when it was in the combined `spring-boot-autoconfigure` dependency. Now it resides in `spring-boot-mongodb` that is not added to the applications anymore.
- The `FreeMarkerAutoConfiguration` had to be excluded when it was in the combined `spring-boot-autoconfigure` dependency. Now it resides in `spring-boot-freemarker` that is not added to the applications anymore.
- The `DataSourceAutoConfiguration` had to be excluded when it was in the combined `spring-boot-autoconfigure` dependency. Now it resides in `spring-boot-jdbc`. That dependency is still needed in the Studio Server and User Changes applications (along with the exclusion of the `DataSourceAutoConfiguration`). But it has been removed in all other applications.

The former `ServletWebServerFactoryAutoConfiguration` and `EmbeddedWebServerFactoryCustomizerAutoConfiguration` contained configurations for various supported web servers. The configurations for each web server have now been moved to a focused module JAR. In case of Tomcat, the auto configuration class is now named `TomcatServletWebServerAutoConfiguration` and resides in the `spring-boot-tomcat` module JAR.

Upgrading Jackson

Spring Boot now uses Jackson 3 as its preferred JSON library. But for projects that cannot easily migrate, Jackson 2 is still supported (until excluding Spring Boot 4.2). Jackson 3 uses new group IDs and package names with `com.fasterxml.jackson` becoming `tools.jackson`. Although the coordinates of both Jackson versions are disjunct, it turned out that Spring's Jackson 2 and 3 integrations produce various conflicts when the dependencies of both versions are on the classpath of the CoreMedia applications. As migrating

the Jackson integrations of the CoreMedia CMS is a rather complex and time-consuming task, Jackson 3 dependencies have been avoided for now.

The migration to Jackson 3 is planned for the AEPs that will be released in spring or autumn 2026. If possible, upcoming AMPs may support both Jackson versions on the classpaths, but this requires further investigation.

Upgrading Testing Features

Mocks or test clients may not be working like before. Test classes may require additional annotations now:

- @AutoConfigureMockMvc
- @AutoConfigureWebTestClient
- @ExtendWith(MockitoExtension.class)

In some cases, when @Mock and @MockitoBean are both used in the same test class, the @Mock annotation may have to be replaced with the static Mockito#mock(Class<T>) invocation.

Public API Changes

Spring Boot's new modular design required changes to the com.coremedia.cms.delivery.configuration.DeliveryPropertiesAutoConfiguration:

- The class annotations changed from @AutoConfiguration(after={ServletWebServerFactoryAutoConfiguration.class,WebMvcAutoConfiguration.class}) @EnableConfigurationProperties(DeliveryConfigurationProperties.class) to @AutoConfiguration(after=WebMvcAutoConfiguration.class) @EnableConfigurationProperties({DeliveryConfigurationProperties.class,ServerProperties.class,WebMvcProperties.class}).
- The bean method changed from deliveryBaseURI(DeliveryConfigurationProperties, ObjectProvider<ServerProperties>, ObjectProvider<WebMvcProperties>) to deliveryBaseURI(DeliveryConfigurationProperties, ServerProperties, WebMvcProperties).

7.1.2 Updated Third-Party Libraries

Apart of the updates that have been mentioned in the previous sections, many further libraries have been updated. Updating dependencies is a continuous task, each CoreMedia release includes several updates to third-party libraries. As this upgrade guide is not tied to a particular CMCC 13 version, it does not make sense to list all minor or patch version updates here. Please refer to the CMCC 13 Release Notes for the CMCC 13 versions of interest.

7.2 API Changes

This section contains changes in the CoreMedia APIs. You will find more API changes in the component specific subsections.

7.2.1 Support for Personalization Variants

In order to support personalization variants, a new meta-property `variant` was added to Content items. It is immutable once set during content creation. It is accessible through `isVariant()` and its inverse `isBaseline()`. For a baseline content item, multiple variants may be created. Variants are created with the same content type as their baseline. They are used for delivering A/B tests or specialized content for user segments.

For custom code, please check whether it can handle variant content items. Especially, check whether variants that are returned from methods like `getReferrersWithDescriptor(...)` break the analysis of complex content structures. In such cases, you can use the new methods `getBaselineReferrersWithDescriptor(...)` to avoid reaching variants accidentally, as these methods efficiently limit the returned collection to baseline content items. Similarly, instead of using `getContentsWithValue(...)` it might more robust to access observed properties using `getBaselineContentsWithValue(...)`. Similarly, the already deprecated class `ContentStringPropertyIndex` might not be appropriate, because it includes variants in its results.

For the sites service and its associated classes, personalization variants are generally ignored in returned collections, but accepted as input arguments. That is, you can use the sites service for determining the site of a variant or for deriving variants, but variants are not contained in methods like `ContentSiteAspect.getVariantsByLocale`. In the unlikely case that you need to include variants in this context, use the alternative methods like `ContentSiteAspect.getAllVariantsByLocale` or `ContentSiteAspect.getAllVariants`.

The variant feature only becomes accessible to editors if you install the Native Personalization plugin for Studio. Without the plugin, editors cannot create variants. Before doing that, all Content Servers must have been upgraded to a release that supports variants.

7.2.2 Workflow Client API

The old workflow client API and its implementations have been removed. They were actually unused after removing the Site Manager. This API was deprecated with 2406.2.2 and 2506.0.2. This affects the classes in the following packages:

- `com.coremedia.workflow`
- `com.coremedia.workflow.common.util`
- `com.coremedia.workflow.worklist`

7.2.3 Removed Unused Code from UriComponentsHelper

Unused code has been removed from the `com.coremedia.objectserver.web.links.UriComponentsHelper`. This results in removed and renamed public methods.

Renamed Methods

- `appendOrOverride(UriComponents, UriComponents)` → `append(UriComponents, UriComponents)`

Removed Methods

- `appendOrOverride(UriComponents, UriComponents, boolean)`
- `override(UriComponents, UriComponents)`
- `prependPath(String, UriComponents)`
- `prependPath(UriComponents, UriComponents)`
- `prependPath2(String, UriComponents))`

See UriComponentsHelper for details.

7.2.4 Removed Unused Legacy CORS Configuration Properties

The deprecated and unused *`cae.cors.*`* (CAE) and *`caas.cors.*`* (Headless Server) configuration properties have been removed. Since CMCC-12 release

2506.0.0, these delivery-specific CORS configuration properties were unused as they were replaced by the generalized *cors.** configuration properties.

See the CORS Properties reference in the Deployment Manual for details.

7.2.5 Minor API Change in WorkflowObject

The return type of `com.coremedia.cap.workflow.WorkflowObject#getWarnings` has been weakened from `List<WorkflowException>` to `List<CapException>`. Due to a CMS internal generics inconsistency the result list of `getWarnings` may contain `CapException` objects even in older CMS versions. This fix does not change the behavior, but only clarifies the type. Custom code may have to be adjusted accordingly.

7.2.6 Public Workflow Server API has Moved

Public workflow server API has been moved. The Java packages of the affected classes didn't change, but they are now contained in Java artifact `com.coremedia.cms:cap-workflow-server` instead of `com.coremedia.cms:cap-workflow`.

See Section 7.3, "Adding gRPC Support for Component Communication" [36] for details.

7.3 Adding gRPC Support for Component Communication

For internal communication between CoreMedia CMCC components, gRPC has been added as an alternative to CORBA. Communication via CORBA is still available and the default. As long as you plan to stick with CORBA, no changes to your code or operations procedures are required. Should you want to switch to gRPC as your communications protocol between CMCC components, the following sections will provide information and pointer for additional details to do so.

In the course of adding gRPC as a communications protocol, however, breaking changes were introduced:

- The old workflow client API and its implementations have been removed. They were actually unused after removing the Site Manager. This API was deprecated with 2406.2.2 and 2506.0.2. This affects the classes in the following packages:
  - `com.coremedia.workflow`
  - `com.coremedia.workflow.common.util`
  - `com.coremedia.workflow.worklist`
- The public workflow server API has been moved. The Java packages of the affected classes didn't change, but they are now contained in Java artifact `com.coremedia.cms:cap-workflow-server` instead of `com.coremedia.cms:cap-workflow`. Consequently, they cannot be used in extensions other than for the workflow server anymore. For clients, use the Unified API. Documentation for the moved classes used to be at CoreMedia Shared Common API and can now be found at CoreMedia Workflow Server API.

Upgrade Information

NOTE

With release 2512.0 of the CoreMedia system, the Workflow Server does not yet offer gRPC services to its clients. Activating gRPC on clients (for example, Studio Server or command line tools) won't do any harm as long as these are of release 2512.0, too. Future clients, though, will fail to connect to that Workflow Server if gRPC is active on them.

Keep this in mind when operating a mixed-release CoreMedia system setup. In such scenarios (for example, during upgrades), it is recommended to leave clients in their default protocol mode CORBA.

While servers always offer both protocols for clients to connect, clients can be configured to use either gRPC or CORBA. The protocol can be chosen freely on each start of a component. Currently, CORBA is the default protocol.

Note that Content Servers and the Workflow Server also contain client parts for internal communication, for example, for internal repository access, publication or replication. Choosing a protocol on a server therefore does make a difference although services offered by the servers to their clients always cover both protocols.

Information for Developers

See Section 3.2.2, "Docker Compose Setup" in *Blueprint Developer Manual* , Section 4.3, "Build and Run the Applications" in *Blueprint Developer Manual* on how to activate gRPC in a development setup. Refer to Unified API Developer Manual, Section 4.1.1, "Creating a Connection" in *Unified API Developer Manual* for details on connection parameters for gRPC.

Information for Operations Engineers

See Section 4.4, "Communication between the System Applications" in *Operations Basics* for details on how to configure and secure gRPC communication in your environment.

Replicator and Publisher require specific configuration for gRPC:

- For the Publisher, refer to Section 3.2.3, "Properties for the Publisher" in *Deployment Manual*.
- or the Replicator, refer to, Section 3.2.5, "Properties for Replicator Configuration" in *Deployment Manual*.

In a Nutshell

Adjust hostnames and ports in the following variables to your setup where necessary. Mind that there is no scheme to the gRPC addresses as those are not URIs.

To enable gRPC communication for the Content Management Server, set the following environment variables on the Content Management Server process:

- *REPOSITORY_USEGRPC=true*
- *SPRING_GRPC_CLIENT_CHANNELS_CAP_ADDRESS=content-management-server:40165*
- *SPRING_GRPC_CLIENT_CHANNELS_WF_ADDRESS=workflow-server:40365* (not for Workflow Server; this setting is not in effect yet but may be added without harm)
- *REPOSITORY_HTTPBASEURI=http://content-management-server:40180*

To enable gRPC communication for a Replication Live Server, set the following environment variables on the Replication Live Server process:

- *REPOSITORY_USEGRPC=true*
- *PUBLISHER_TARGET_0_GRPC_ENDPOINT=master-live-server:40265* (repeat for each publication target)
- *PUBLISHER_TARGET_0_HTTP_BASE_URI=http://master-live-server:40280* (repeat for each publication target)

To enable gRPC communication for a Master Live Server, just set

- *REPOSITORY_USEGRPC=true*
- *SPRING_GRPC_CLIENT_CHANNELS_CAP_ADDRESS=master-live-server:40165*
- *REPLICATOR_PUBLICATION_HTTP_BASE_URI=http://master-live-server:40280*

To enable gRPC communication for the command line tool publishall, see the new command line options in Section 3.13.3.5, “Publishall” in *Content Server Manual*.

7.4 Studio Client Changes

This section describes the changes in the user interface of CoreMedia Studio and in the build of Studio client.

7.4.1 ExtJs Workflow App Replaced by React Workflow App

As part of our ongoing modernization efforts, the ExtJs-based Workflow App has been replaced by a new React-based Workflow App. The new app provides an improved user experience and better performance, while maintaining the same functionality as the previous version.

7.4.1.1 Public API changes (ExtJS)

With the removal of the ExtJS Workflow App the following ExtJS components and plugins have been removed. Neither of the classes were in use by the content app so if you have still used any of these classes this might just be dead code.

Package @coremedia/studio-client.ext.control-room-editor-components

- **Removal**
  - `AddInboxDetailFormPlugin`
  - `AddProcessDetailFormPlugin`
  - `AddFinishedProcessFormPlugin`
  - `WorkflowDateDisplayField`
  - `WorkflowTextDisplayField`
  - `WorkflowForm` / `WorkflowFormBase`
  - `DefaultPublicationWorkflowDetailForm` / `DefaultPublicationWorkflowDetailFormBase`
  - `DefaultPublicationWorkflowInfoForm` / `DefaultPublicationWorkflowInfoFormBase`
  - `DefaultTranslationWorkflowDetailForm` / `DefaultTranslationWorkflowDetailFormBase`
  - `DefaultTranslationWorkflowInfoForm` / `DefaultTranslationWorkflowFormBase`
  - `EmbedWorkflowObjectFromContentAppAction`

  - `WorkflowObjectFromContentAppAction`
- Removed the following config from `AddWorkflowPlugin`, `AddPublicationWorkflowPlugin` and `AddTranslationWorkflowPlugin`
  - `inboxForm`
  - `pendingForm`
  - `finishedForm`
- The module `PublicationWorkflowConstants` is now deprecated. Utilize the `PublicationWorkflowConstants` class of `@coremedia/studio-client.workflow-models` instead.

@coremedia/studio-client.ext.workflow-components

We removed the `errorIssueExists` method from the class `WorkflowIssuesUtil`

Package @coremedia/studio-client.workflow-services-api

We removed the `WorkflowRouteKeys` class.

7.4.1.2 Custom Workflow API Changes

While the Workflow App has been replaced by a new implementation based on React instead of ExtJs, the API for custom workflows has largely been kept unchanged. However, there are notable breaking changes:

Localization Based on Resource Bundles

The well-known Studio mechanism for localization based on resource bundles was (and in the Content App still is) based on ExtJs mechanisms. So to localize your custom workflow labels and texts, you need to apply a new mechanism. You still have resource bundles as before in different languages. However, you need to register the localized bundles for "de" and "ja" using the `registerLocale()` function:

```
import { registerLocale } from "@coremedia/studio-client.i18n-models";

registerLocale(Gcc_properties, "de", async () => {
  await import("./Gcc_de_properties");
});
```

Following that, when localizing a specific resource bundle key, you need to use the `getLocalizer()` function:

```
import { getLocalizer} from "@coremedia/studio-client.i18n-models";

const localize = await getLocalizer(Gcc_properties);
```

```
TextField({
  label: localize("TranslationGlobalLink_submission_status_key"),
  ...
}),
```

WorkflowPlugin API

The type `WorkflowPlugin` is no longer generic, this is now only the case for its properties *`startWorkflowFormExtension`* and *`runningWorkflow-FormExtension`*. In addition, for the two form extension, the factory function `StartWorkflowFormExtension<ViewModel>()` and `RunningWorkflowFormExtension<ViewModel>()` have been introduced respectively. So the registration of a new workflow plugin follows this pattern:

```
interface GccViewModel {
 ...
}

const getGccWorkflowPlugin = async (): Promise<TranslationWorkflowPlugin> =>
 {
  workflowType: "TRANSLATION",
  workflowName: "TranslationGlobalLink",
  ...
  startWorkflowFormExtension: StartWorkflowFormExtension<GccViewModel>({ ...
 }),
  runningWorkflowFormExtension: RunningWorkflowFormExtension<GccViewModel>({
 ... }),
  ...
}

getGccWorkflowPlugin().then((gccWorkflowPlugin) => {
  workflowPlugins._.addTranslationWorkflowPlugin(gccWorkflowPlugin);
});
```

For more details, see Section 9.28, "Custom Workflows" in *Studio Developer Manual*.

ResourceBundleUtil

The utility class `ResourceBundleUtil` from `@jangaroo/runtime/l10n/ResourceBundleUtil` has been deprecated in `jangaroo.npm` and is not supported inside the workflow app. To replace `ResourceBundleUtil.override` please just utilize Web API `Object.assign` instead which doesn't require any import. You can also remove the dependency `@jangaroo/runtime` from your package if it was only utilized for that.

To retain code completion and type errors if an overridden resource bundle key does no longer exist in the default resource bundle, you can utilize `satisfies Partial<MyResourceBundle_properties>` (supported by jangaroo.npm 3.3.1+), for example,

```
import ResourceBundleUtil from "@jangaroo/runtime/l10n/ResourceBundleUtil";
import MyResourceBundle_properties from "./MyResourceBundle_properties";

ResourceBundleUtil.override(MyResourceBundle_properties, {
  someKey: "...",
});
```

should be replaced by:

```
import MyResourceBundle_properties from "./MyResourceBundle_properties";

Object.assign(MyResourceBundle_properties, {
  someKey: "...",
} satisfies Partial<MyResourceBundle_properties>);
```

Public API

Whenever utilizing studio-client API you now need to make sure to only utilize public API. In addition, public API needs to be imported from the package export, meaning that deep path imports inside the package will not work. The new plugin build will report corresponding warnings and errors.

Example:

```
import contentTypeLocalizationRegistry from
"@coremedia/studio-client.cap-base-models/content/contentTypeLocalizationRegistry";
```

now needs to be

```
import { contentTypeLocalizationRegistry } from
"@coremedia/studio-client.cap-base-models";
```

The identifiers are always named after the module file name of the old import.

If you have split your plugin into multiple packages this applies to all packages used inside the runtime code of the plugin. Also be aware that in case you have used packages that did not contain any public API at all, the plugin build may not report any warnings or errors. Those packages will be handled as custom third party packages, so their code is just bundled into your plugin. This could lead to unintended side effects if you rely on shared runtime state or similar in that case.

Extension Mechanism

While the API is mostly the same, there are changes regarding affecting the extension package structure because the new Workflow App is no longer running with ExtJS and though cannot work with the ExtJS based “Dynamic Packages” mechanism.

This has been replaced with a new mechanism based on Module Federation.

package.json Changes

In order to switch an extension for the old workflow app to the new workflow app, you need to make notable changes to the devDependencies and the scripts of the package.json:

1. Declare the package type as “module” by adding:

```
"type": "module",
```

2. Remove “@jangaroo/core” and “@jangaroo/build” and make sure you have at least these devDependencies

```
"devDependencies": {
  "@babel/core": "^7.23.2",
  "@coremedia/eslint-config-studio-client": "2506.0.0",
  "@coremedia/studio-client.build-config": "2506.0.0",
  "@coremedia/studio-client.workflow.shared-modules": "2506.0.0",
  "eslint": "^8.57.0",
  "rimraf": "^5.0.7",
  "typescript": "^5.8.2",
  "webpack": "^5.98.0",
  "webpack-cli": "^6.0.1"
}
```

3. Make sure to have at least these scripts:

```
"scripts": {
  "clean": "rimraf ./dist",
  "prebuild": "sync-tsconfig-project-references src
--includeDevDependencies",
  "build": "tsc --project src && webpack build --mode=production",
  "watch": "webpack watch --mode=development",
  "lint": "eslint \"src/**/*.ts*\""
}
```

4. Add a new section “coremedia” with the following entry (replacing the $placeholder with a name containing only letters or underscore [a-z_]+):

```
"coremedia": {
  "projectExtensionFor": "studio-client.workflow",
  "plugins": {
    "$placeholder": "./dist/appEntry.js"
  }
},
```

Other Files

1. Remove `jangaroo.config.js`
2. Create `.eslintrc.cjs`

```
module.exports = {
  extends: ["@coremedia/eslint-config-studio-client"],
};
```

3. Create `babel.config.js`

```
import { babelConfig } from "@coremedia/studio-client.build-config";

export default babelConfig;
```

4. Create `webpack.config.js`

```
import { sharedModules } from
"@coremedia/studio-client.workflow.shared-modules";
import { createWorkflowPluginConfig } from
"@coremedia/studio-client.build-config";

export default createWorkflowPluginConfig({
  name: "$placeholder",
  sharedModules,
});
```

   Where $placeholder must be the same identifier that you have used in your `package.json` file.

5. Create `src/index.ts`

```
// empty, must be there for webpack...
```

6. Create `src/tsconfig.json` (the new prebuild script will automatically modify this `tsconfig.json` later):

```
{
  "extends": "@coremedia/studio-client.build-config/tsconfig-shared.json",

  "compilerOptions": {
    "rootDir": ".",
    "outDir": "../dist/src"
  }
}
```

7. Create `src/plugin.ts`

```
export const initPlugin = async () => {
  // add your plugin code here, e.g. start utilizing the custom workflow
api
};
```

Now you have a base skeleton for the new plugin mechanism. You can start adding your own code to the `src/plugin.ts` file.

Migrating Content Type Localization for Content App and Workflow App

You usually want to have Content Type Localization only once in a shared packages. If you have sticked to the examples for Content Type Localization provided in the Blueprint, there is a migration path for the new plugin mechanism. While all examples in the Blueprint are adjusted, the general approach looks as follows.

Add the initialization code to the `src/plugin.ts` file of your plugin.

```
export const initPlugin = async () => {
  const module = await
import("@coremedia-blueprint/studio-client.my-doctypes");
  await module.initMyDocTypes();
};
```

For the shared package itself (in the example this is `@coremedia-blueprint/studio-client.my-doctypes`) you now need to create and export the corresponding `init` method. This can be as simple as just exporting it in their `index.ts` file:

```
import { contentTypeLocalizationRegistry } from
"@coremedia/studio-client.cap-base-models";
import MyDocTypes_properties from "./MyDocTypes_properties";

export const initMyDocTypes = async (): Promise<void> => {
  contentTypeLocalizationRegistry.addLocalization(MyDocTypes_properties);
};
```

For more complex examples (including different locales), check the packages `@coremedia-blueprint/studio-client.workflow.blueprint-doctypes-plugin` and `@coremedia-blueprint/studio-client.blueprint-doctypes` in the Blueprint workspace.

Customization via ExtJS Plugins

The ExtJS Plugin mechanism was never supported in the old Workflow App. However, it was possible to ignore that and just utilize the API in order to make customizations. As we switched to React, there is no replacement for that and though we cannot offer a migration path here.

Customization for the Workflow App can now only be done via the provided `CustomWorkflowApi` and additional APIs like the `contentLocalizationRegistry`.

7.4.1.3 Changed LinkList Thumbnails

The LinkList component of Studio has the content type icon as an overlay in the lower right corner now. This icon replaces the type icon column which has been removed from all Blueprint usages. The former thumbnail column (without type icon) is still available, but has been deprecated. In addition, the distance between the thumbnail and the content item name has been increased to make the view more appealing.

7.4.1.4 Include Updated Referenced Content in Publication Workflows

When starting a publication workflow in Studio there is now a checkbox to include updated referenced content - similar when starting a translation workflow. `PublicationWorkflowPlugin` and `DefaultStartPublicationWorkflowFormBase` also have the config parameter *`hideDependentContentsStrategyChooser`* which can be set to hide the checkbox.

7.4.2 Technical Improvements and Dependency Updates

This section summarizes technical improvements, dependency updates, and related internal adjustments included in this release, relevant for Studio Client.

7.4.2.1 Public API changes for Localization / Severity

The following types / consts / classes are now deprecated. You can find additional information about what to use instead in the code documentation:

- `Severity` of `@coremedia/studio-client.base-models`
- `ILocalization`, `ILocalizationWithIcon`, `ITypeLocalization`, `IPropertyLocalization` and `IContentTypeLocalization` of `@coremedia/studio-client.cap-base-models`
- `IContentTypeLocalizationRegistry#getLocalizationByContentType` of `@coremedia/studio-client.cap-base-models`

For the interfaces `WorkflowIssueLocalization`, `WorkflowIssuesLocalization` and `WorkflowLocalization` of `@coremedia/studio-client.workflow-plugin-models` we now have explicit configuration interfaces `WorkflowIssueLocalizationConfig`, `WorkflowIssuesLocalizationConfig` and `WorkflowLocalizationConfig`.

While the former is meant to provide consistent, more strict access to registered Localization, the latter is meant to provide configuration where some properties can be undefined or have shorthand syntax. The configuration interfaces are now also used when registering new workflows via the CustomWorkflow API.

This means two things:

- In case you have made use of the non-configuration interfaces inside your custom code, e.g., in function signatures or to store local variables, you might need to switch to the new configuration interfaces instead.
- You might be able to simplify code that utilizes localization interfaces, as many properties can no longer be undefined and have their shorthand syntax resolved to the explicit syntax.

7.4.2.2 Third Party Updates

The version of many third party dependencies was updated to their latest minor/patch versions.

While we expect no problems with these updates, there is one notable change after updating the ESLint dependencies regarding the format:

If the `extends` fragment contains more than one base class and the line is too long, the format of this multiline `extends` fragment changed. Now each base class is written on a new line.

The affected Studio-Client source code was updated accordingly.

7.4.2.3 Node.js 24 and PNPM 10.24.0 Upgrade

Node was upgraded to the latest LTS version 24.11.1.

PNPM has received a minor upgrade to 10.24.0.

7.4.2.4 Removed Animations

The .webm based Studio animations have been removed from the Studio and replaced by .svg files.

All searches and empty states are affected by this, including the workflow app.

When you have used the animations for your own customizations, you have to switch to the svg files.

7.4.2.5 Removed Tab-Expand

Removed Tab-Expand to reduce vertical scrolling in Content forms by smarter usage of screen real-estate.

The configuration flag for the tab expand has been removed. All formular tabs are always shown now.

The TabExpandPlugin plugin has been removed too.

The design of the tabs has been changed to a button style.

7.4.2.6 CKEditor5 has been updated to version 46.1.1

CKEditor5 has been updated to version 46.1.1 and may require an update of the ckeditor5 dependency in custom plugins.

The latest release of CKEditor 5 contains various improvements but also a lot of breaking changes.

See https://github.com/ckeditor/ckeditor5/blob/master/CHANGELOG.md

While updating to the intermediate version 43.3.1, the module struture of CKEditor has been changed to unified package.

Migration

If any imports like `@ckeditor/ckeditor5-*` are used, migration efforts are necessary, as only imports from `ckeditor5` are allowed, all other imports have to be replaced by imports from `ckeditor5`. If imports are not availabe, utilize workarounds like alias configurations for webpack

See the CKEditor5 release notes for details.

NOTE

Due to inconsistent package exports in some CKEditor5 plugins, you might be forced to create a patch and add a default export to the `package.json` file of the plugin package. Please see the PNPM patch documentation to learn about how to patch packages or the CoreMedia CKEditor5 plugins workspace's `package.json` file to see how the patch is applied.

7.4.2.7 Studio-Client packages for build-config, eslint-config and i18n-models now have their own release cycle

The versions of following packages are no longer matching the CoreMedia CMS Release version but have (similar to jangaroo packages) a dedicated release cycle:

- `@coremedia/studio-client.build-config`
- `@coremedia/studio-client.eslint-config` (old name: `@coremedia/eslint-config-studio-client`)
- `@coremedia/studio-client.i18n-models`

In case you are referring these packages without the "catalog:" protocol in your package.json files make sure to utilize the corresponding version mentioned in the catalog entry of apps/studio-client/pnpm-workspace.yaml.

7.4.2.8 Studio Client Workspace now employs the PNPM Catalog Protocol

The Studio Client Workspace now employs the usage of the PNPM catalog protocol for all dependencies in package.json of all packages. Using the catalog protocol makes it much easier and less error prone to maintain only one version per dependency in a workspace. For details refer to the original documentation at Catalogs | pnpm.

Changes:

- pnpm-workspace.yaml has been adjusted to manage the catalog via the corresponding entry.
- catalogMode is set to prefer (see: Catalogs | pnpm).
- All package.json files have been adjusted so external (non-workspace) dependencies are using the catalog protocol. This applies to:
  - dependencies
  - devDependencies
  - peerDependencies

This means that dependencies inside all package.json files will either have “workspace:*” or “catalog:” as version.

Upgrade Notes:

- If you have added new external dependencies to packages, we highly suggest to add them to the catalog as well.
- Adjust added dependencies to existing package.json files accordingly.
- Adjust your dependencies in newly added package.json files accordingly.

7.4.2.9 Public API changes for studio-client packages

The following changes to Public API packages have been made:

Package @coremedia/studio-client.base-models

Deprecations

The module `TimeZones_properties` has been deprecated. Use the new `timeZoneLocalizationRegistry` to add localization for time zones instead.

Package @coremedia/studio-client.cap-base-models

Changes

- Member `svgIcon` of `IContentTypeLocalization` is now `string | URL`, was `string`.
- `getPreferencesState` of `PostUploadPreferences` now returns a `Promise<boolean | undefined>`. In addition to this the type has been sharpened, reflecting that it can also return undefined (which it could before but it was not stated).

Deprecations

The import for types `ILocalization` and `ILocalizationWithIcon` have been deprecated. Please import the types now from `@coremedia/studio-client.base-models`.

Package @coremedia/studio-client.content-services-api

Deprecations

The methods `getActiveContent` and `getOpenedContents` of `ContentFormService` have been deprecated. Please utilize `observe_activeContent` and `observe_openedContents` instead (also check https://rxjs.dev/api/index/function/firstValueFrom).

Package @coremedia/studio-client.interaction-services-api

Removal

The abstract classes `Clipboard` and `ClipboardItem` have been removed in favor of the standard Web API interface (Clipboard API - Web APIs | MDN).

The class `ClipboardItemImpl` and the singleton `clipboard` have been removed without replacement. To access clipboard data utilize new API `observeClipboardData` and `writeToClipboard` of new package `@coremedia/studio-client.interaction-services-toolkit` instead.

Move

`DataTransferTypes` has been moved to package `@coremedia/studio-client.interaction-services-toolkit`.

Package @coremedia/studio-client.project-services-api

Deprecations

The methods `getActiveProject` and `getOpenedProjects` of `ProjectFormService` have been deprecated. Please utilize `observe_activeProject` and `observe_openedProjects` instead (also check https://rxjs.dev/api/index/function/firstValueFrom).

Package @coremedia/studio-client.workflow-services-api

Deprecations

The methods `getActiveProcess`, `getActiveTask`, `getOpenedProcesses` and `getOpenedTasks` of `WorkflowFormService` have been deprecated. Please utilize `observe_activeProcess`, `observe_activeTask`, `observe_openedProcesses` and `observe_openedTasks` instead (also check https://rxjs.dev/api/index/function/firstValueFrom).

General

The API of many types extending (and including) `RemoteBean` has been sharpened. It is now correctly stated if a property can be null. In addition to this all properties that are only available if a `RemoteBean` has been loaded are now also declared as undefined. Furthermore, the `isLoaded` method has been improved with a type guard meaning that Typescript is now working when checking `isLoaded` first (see TypeScript Narrowing) before accessing a `RemoteBean` property.

While improving the types has no impact on runtime code, you might need to consider the undefined / null results in your code in case you have enabled strict null checks in your TypeScript project. The latter however, is not relevant for packages build with jangaroo as strict null checks are not enabled per default.

7.4.2.11 CoreMedia CKEditor5 Plugins updated to Version 23.0.0

This update comes with a minor breaking change: Empty `xlink` attributes will no longer be stripped from CKEditor Rich Text.

7.4.2.12 CKEditor5 updated to version 45.2.0

CKEditor5 has been updated to version 45.2.0 and comes with updated versions of CoreMedia's CKEditor5 plugins. The update includes 2 major changes:

- Icons are no longer imported as a single `icons` class, but need to be imported separately.
- The link balloon of the CKEditor's link plugin has been re-implemented completely and now consists of a `ToolbarView` instead of the `LinkActionsView`. CoreMedia's link plugin extension has also been updated to be compatible.

If you have customized the link targets for the link balloon, you will now have to add those targets to the `link.toolbar` configuration. Please have a look at the corresponding documentation.

Migration Path

Previously you would have configured the link targets via this configuration:

```
//...
link {
  //...
  targets: [
    "_self",
    {
      name: "_parent",
      icon: parentIcon,
    },
  ]
}
```

Now, the new `toolbar` configuration handles which targets are displayed inside the link balloon. You will still need to define custom targets in `targets` to remove the default targets ("_self", "_new", "_other", "_top") and simply add them to the `toolbar`. Please note, that you will also need to add the buttons to display, edit or unlink the link here as well:

```
//...
link {
  //...
  toolbar: ["linkPreview", "editLink", "|", "_self", "_parent", "|", "unlink"],

  targets: [
```

```
    {
      name: "_parent",
      icon: parentIcon,
    },
  ]
}
```

The default toolbar looks like this. Please feel free to reconfigure it in `ckeditorDefault.ts` if you previously changed the link targets:

```
["linkPreview", "editLink", "|", "_self", "_blank", "_embed", "_other", "|",
 "unlink"]
```

7.4.2.13 Header Toolbar Revamp

The header toolbar has been reorganized. Since the separators and the buttons might have been used as reference for adding other elements, this change is considered breaking. In detail, the following changes were done:

- The library, control room and quick search buttons have been moved from the right side to the left.
- The upload button has been removed from the dashboard and added to the header toolbar directly.
- The bookmarks have been moved into the new header toolbar menu "Open". This menu contain additional menu items for the control room and library too.
- All separators have been removed from the header toolbar.
- The saved searches have been moved into the new header toolbar menu **Open**.

7.4.2.14 Switched to pnpm Workspace Protocol

Since pnpm 9 the usage of the workspace protocol is the default configuration in pnpm workspaces (see Release v0.0.0 - pnpm - new default “false” for link-workspace-packages). While CMCC 2406.0.0 already uses pnpm 9 the releases did arrive too shortly after each other to fully adapt the studio-client workspace to this change. That’s why we enabled linking of workspace packages again via an `.npmrc` file.

For this release we changed our dependencies and tooling around the dependencies (like the extensions mechanism) to support and utilize the workspace protocol. For this all dependencies to packages within the studio-client workspace have been adjusted in their corresponding `package.json` files.

Dependencies

If you have any customizations to dependencies of existing packages or have added new packages, make sure that you utilize the workspace protocol as well. This means that in the `package.json` file of the corresponding packages all entries inside "dependencies", "devDependencies" or "peerDependencies" need to be adjusted if they reference another package inside the workspace from "1.0.0-SNAPSHOT" (default version in the Blueprint workspace, might be different) to "workspace:*".

Important: Please do not change dependencies referencing released artifacts starting with "@coremedia/" or any third-party vendor.

Changes to Provided Publish Task

> NOTE
> In case you do not release any npm package from within the studio-client workspace, you can skip this section.

With this change we also adjusted the provided publish task and configuration to no longer utilize "@jangaroo/publish" but the default "pnpm publish". This means that the dependencies to "@jangaroo/publish" inside the "devDependencies" section of every package have been removed.

In addition to this we also removed the "publish" script for every package where it was no longer necessary.

Before the change, we prevented using "pnpm publish" by making the packages "private" via the corresponding entry in their `package.json` file. This has been removed as well.

Finally, the publish-task provided inside `Dockerfile.tasks` has been adjusted accordingly.

7.4.2.15 Removed Usages of jangaroo-net and Corresponding URI class

We removed the usage of `jangaroo-net` and its corresponding `URI` class in the studio-client to streamline the API and get rid of obsolete dependencies.

The `package.json` file of the Blueprint package `@coremedia-blueprint/studio-client.main.lc-studio` was adjusted accordingly and the implementation of `src/action/OpenCreateExternalPageDialogActionBase.ts` has slightly changed.

In addition for the package `@coremedia/studio-client.main.editor-components` the `getUri` method of the `PreviewURI` class has been removed. Please use `getUrl` instead.

The following non-public API are affected as well:

- `@coremedia/studio-client.client-core`
  - `remoteService`
  - `UrlUtil`
- `>@coremedia/studio-client.client-core-impl`
  - `RemoteService`
- `@coremedia/studio-client.client-core-test-helper`
  - `MockFetch`

7.4.2.16 Public API changed

Public API Removals

- `@coremedia/studio-client.cap-rest-client`
  - `CapList`, `CapListRepository` and `EncryptionService` are not public anymore.
  - `CapConnection`: removed members `getCapListRepository` and `getEncryptionService` (were already marked as @private)

New Public API

- `@coremedia/studio-client.cap-rest-client`
  - `registerMyEditedContentFilter`: allows registering filters for the contents appearing in “My Edited Content”
  - `observeMyEditedContents`: allows observing the filtered contents of “My Edited Content”

Blueprint Adjustments

- `@coremedia/studio-client.cap-rest-client`
  - `ShowAssetsInEditedContentsCheckboxBase` now uses (new) `registerEditedContentsFilter`

7.4.2.17 Upgraded eslint to v9

See Section 7.4.2.10, "Deprecated @coremedia/studio-client.client-core-impl and @coremedia/studio-client.cap-rest-client-impl" [?] for details.

7.4.2.18 Moved Preview Related Entities to cap-rest-client

Preview related entities have been moved from the packages `@coremedia/studio-client.client-core` and `@coremedia/studio-client.client-core-impl` to `@coremedia/studio-client.cap-rest-client` and `@coremedia/studio-client.cap-rest-client-impl`. Please adjust the following import statements:

```
import Previewable from
"@coremedia/studio-client.client-core/data/Previewable";
import PreviewConfig from
"@coremedia/studio-client.client-core/data/PreviewConfig";
import Previews from "@coremedia/studio-client.client-core/data/Previews";
import PreviewsImpl from
"@coremedia/studio-client.client-core-impl/data/impl/PreviewsImpl";
```

to

```
import { Previewable } from "@coremedia/studio-client.cap-rest-client";
import { PreviewConfig } from "@coremedia/studio-client.cap-rest-client";
import { Previews } from "@coremedia/studio-client.cap-rest-client";
import { PreviewsImpl } from "@coremedia/studio-client.cap-rest-client-impl";
```

7.4.2.19 studio-client Container is now read-only

The studio-client container is now read-only in the Docker compose setup. The following volumes were added for the nginx and plugin setup:

- `/usr/share/nginx`
- `/etc/nginx/conf.d`
- `/var/cache/nginx`
- `/var/run`

7.5 Content Server Changes

This section describes changes applied to the different content servers.

7.5.1 Session Check for Blob Download from Content Server can be Disabled

Blob Downloads from Content Servers are verified for a valid login session on each call. With the new configuration option `cap.server.blob-download-session-validation`, this session validation can be disabled. The default for the option is true, that is, each download request is verified for a valid login session. If set to false, the session validation for downloads is disabled. This is useful on Replication Live Servers to allow for load-balanced blob download. It must only be set to false if corresponding servers are appropriately secured by firewalls, private networks, or the like. Otherwise, public unsecured read access to blobs could be possible. Blob Uploads are always secured by session validation, regardless of the option's setting.

7.6 Content Application Engine Changes

This section describes changes applied to the Content Application Engine (CAE).

7.6.1 Removed Unused Legacy CORS Configuration Properties

The deprecated and unused `cae.cors.*` (CAE) and `caas.cors.*` (Headless Server) configuration properties have been removed.

Since CMCC-12 release 2506.0.0, these delivery-specific CORS configuration properties were unused as they were replaced by the generalized `cors.*` configuration properties. Please see ???? for details.

7.7 Headless Server Changes

This chapter contains detailed information about necessary upgrade steps for the Headless Server.

7.7.1 Removed Unused Legacy CORS Configuration Properties

The deprecated and unused `cae.cors.*` (CAE) and `caas.cors.*` (Headless Server) configuration properties have been removed.

Since CMCC-12 release 2506.0.0, these delivery-specific CORS configuration properties were unused as they were replaced by the generalized `cors.*` configuration properties. Please see ???? for details.

7.8 Commerce Integration Changes

This section describes changes applied to the different commerce integration components.

7.8.1 Switched to spring-grpc for Commerce Integration

The Commerce Hub Client uses gRPC to communicate with the Commerce Adapters. The gRPC Spring Boot integration was switched from grpc-spring to spring-grpc.

This change causes that properties below *`grpc.server`* and *`grpc.client`* are no longer relevant. Instead, the properties below *`spring.grpc.server`* and *`spring.grpc.client`* must be used respectively.

Along with this change, the commerce hub client setting `com.coremedia.blueprint.base.livecontext.client.settings.CommerceSettings#getEndpoint` was removed. Use `com.coremedia.blueprint.base.livecontext.client.settings.CommerceSettings#getEndpointName` instead and configure an ops-friendly symbolic name, which is then configured as a named gRPC channel below *`spring.grpc.client.channels`*.

More specifically, this means that your commerce struct's endpoint value is no longer picked up to establish a commerce connection. Introduce a String property named *`endpointName`* in your commerce struct and give it the value of a gRPC channel, such as hybris or sfcc.

7.9 Blueprint Changes

This section describes changes in the CoreMedia Blueprint workspace.

7.9.1 Adaptive Personalization (p13n) Extensions Disabled by Default

The Adaptive Personalization extensions have been disabled by default in the Blueprint.

Following extensions have been disabled: `p13n`, `alx-p13n`, `c-p13nl`, `sfmc-p13n`, and `sfmc-es-p13n`. To enable them in your Blueprint, run the following code in you Blueprint workspace:

```
mvn extensions:sync -Denable=p13n,alx-p13n,lc-p13n,sfmc-p13n,sfmc-es-p13n
```

The `p13n-dynamic-include` brick in the Frontend Workspace has been changed to work with CoreMedia's new “Native Personalization” solution and is still used in the example themes.

The Adaptive Personalization extension contains four content types:

- `CMSelectionRules`
- `CMSegment`
- `CMP13NSearch`
- `CMUserProfile`

Instances and references in the example content outside of Adaptive Personalization extensions have been removed. If you, for some reason, have instances of these content types without actually using Adaptive Personalization, you can either destroy these instances, enable the extensions again or just add the content types to your Blueprint.

7.9.2 Elastic Social (es) Extensions Disabled By Default

The Elastic Social extensions have been disabled by default in the Blueprint. This means that after switching to the SQL Persistence for Projects and other editorial features (see Section 4.5.3, “Migration to SQL Persistence” in *Operations Basics* for details), the Blueprint by default does not connect to a MongoDB.

Following extensions have been disabled: `es`, `lc-es`. To enable them in your Blueprint, follow the instructions in Section 4.2.1.2, "Enabling the Elastic Social Extension" in *Blueprint Developer Manual* .

The `es-controlroom` extension has been removed to avoid confusion as it was no Elastic Social Extension. Its functionality has been incorporated into other Blueprint modules.

The `brick-elastic-social` brick in the Frontend Workspace has been left untouched and is still used in the example themes.

The Elastic Social extension contains two content types: `ESDynamicList` and `CMMail`. Instances and references in the example content outside of Elastic Social extensions have been removed. If you, for some reason, have instances of these content types without actually using Elastic Social, you can either destroy these instances, enable the extensions again or just add the content types to your Blueprint.

7.9.3 Removed Analytics Connectors

The integration of Analytics Connectors based on Elastic Core has been removed, that is, extension `es-alx` and parts of extension `alx`, including the analytics content types. Functionality for website tracking is still available and the corresponding documentation has moved to the CoreMedia Blueprint manual.

Removed Features

- Top N Lists including:
  - Analytics content types: `CMALXBaseList (abstract)`, `CMALXPageList`, `CMALXEventList`,
  - Content bean implementations
  - Studio configuration forms
- Retrieval tasks and persistence based on the `ModelService` (module `bp-base-es-alx-common`)
- Studio: generic configuration, performance widgets, deep links

These features are discontinued and will be replaced by an integrated solution.

Removal of the analytics content types (`CMALXPageList` and `CMALXEventList`) leads to problems in the Content-Server, if content of these types exist in the repository. You have two solution options:

To prevent Studio users from creating/editing these content types, you can configure rights appropriately.

- Recommended: Remove all content of type `CMALXPageList` and `CMALXEventList` as described in the documentation at Section 4.3.6, "Deleting Content Types" in *Content Server Manual*.
- If it is not possible to delete the content, you can add reduced analytics content types as custom content types like this:

```
<DocType Name="CMALXBaseList" Parent="CMDynamicList" Abstract="true">
</DocType>
<DocType Name="CMALXPageList" Parent="CMALXBaseList">
</DocType>
<DocType Name="CMALXEventList" Parent="CMALXBaseList">
</DocType>
```

7.10 Frontend Workspace Changes

This section describes changes in the CoreMedia Frontend workspace.

7.10.1 Updated Third-Party Dependencies

Third party dependencies of the frontend workspace have been updated to their latest version fulfilling semantic version ranges.

7.10.2 Frontend Workspace now employs the PNPM Catalog Protocol

The Frontend Workspace now employs the usage of the PNPM catalog protocol for all dependencies in `package.json` of all packages. Using the catalog protocol makes it much easier and less error prone to maintain only one version per dependency in a workspace. For details refer to the original documentation at https://pnpm.io/catalogs.

Changes

- `pnpm-workspace.yaml` has been adjusted to manage the catalog via the corresponding entry
- *`catalogMode`* is set to prefer (see: https://pnpm.io/catalogs#catalogmode)
- All `package.json` files have been adjusted so external (non-workspace) dependencies are using the catalog protocol. This applies to:
  - dependencies
  - devDependencies
  - peerDependencies
- This means that dependencies inside all `package.json` files will either have “workspace:*” or “catalog:” as version.

Upgrade Notes

- If you have added new external dependencies to packages we highly suggest to add them to the catalog as well
- Adjust added dependencies to existing `package.json` files accordingly
- Adjust added dependencies to existing `package.json` files accordingly

7.10.3 Node.js 24 and PNPM 10.24.0 Upgrade

The PNPM lockfile version has been updated to version 8. This means that after the upgrade, the lockfile will be updated to the new format and cannot be used with older versions of PNPM anymore. For details refer to the original documentation at https://pnpm.io/pnpm-lock.

7.10.4 Upgraded eslint to Version v9

Eslint was upgraded to v9. As this involves changing to the new flat config, we decided to provide two functions, which help to set up the config for the corresponding package type:

- `createFrontendEslintConfig`: per default lints "src" and "test" folders with standard rules
- `createFrontendToolsEslintConfig` : per default lints "src" and "test" folders with standard rules and adds additional globals for node

The following steps are necessary for packages using eslint:

1. `package.json`: Remove the file pattern from the eslint script.
2. `eslint.config.(m)js`: Create the eslint.config.mjs file. You can also create a `eslint.config.js` if the package is already ESM package.

   Write the following content into the file, using the corresponding `eslint-config` package

   ```
   import { createFrontendEslintConfig } from
   "@coremedia/eslint-config-frontend";

             export default createFrontendEslintConfig();
   ```

   For more fine granular control over folders that should be linted you can utilize the options parameter of `createEslintConfig`, for example:

   ```
   import { createFrontendEslintConfig } from
   "@coremedia/eslint-config-frontend";

   export default createFrontendEslintConfig({ sourceFolders: ["src", "my-src"]
    });
   ```

3. `.elsintrc`: In case you do not have made any adjustments you can delete the `.eslintrc` file. Otherwise, you need to migrate the changes to v9 (Mi-

grate to v9.x - ESLINT) and add them to the `eslint.config.(m)js` file which can be done like this:

```
import { createEslintConfig } from "@coremedia/studio-client.eslint-config";
import { defineConfig, globalIgnores } from "eslint/config";

export default defineConfig([
  createEslintConfig(),
  {
    // your custom eslint config
  },
]);
```

4. Run the linter and fix reported problems.

As this is a major upgrade of the standard eslint, some rules sets have been added or changed.

CoreMedia's goal was to find the balance between avoiding major breaking changes for the code base but still benefiting from these improvements, so we adjusted the rules if necessary. However, there might still might be some newly reported problems to fix.