

Video

Plug-in User Guide

5.6.0-6

Table of Contents

1 Introduction	4
1.1 About Amazon Elastic Transcoder	4
2 Installation	6
2.1 Conventions	6
2.2 Installing	7
2.3 Re-assembling Applications	7
2.4 Verifying The Installation	8
2.5 Updating The Database Schema	8
2.6 CUE Plug-in Activation	8
3 Configuration	10
3.1 Quick Start	10
3.2 Common Configuration Layer Set-up	11
3.2.1 VideoFileSystemConfiguration.properties	11
3.2.2 KeyframeFileSystemConfiguration.properties	12
3.2.3 Storage.properties	12
3.2.4 DefaultTranscoderConfig.properties	12
3.2.5 TranscodingConfigFactory.properties	13
3.2.6 AWSClientConfig.properties	13
3.2.7 AWSClientService.properties	14
3.2.8 PreviewTranscoderConfig.properties	14
3.2.9 Poller.properties	15
3.2.10 StateChangeNotifier.properties	15
3.3 Application Configuration Layer Set-up	16
3.3.1 AWSMediaInfoGenerator.properties	16
3.3.2 VideoArticleDecorator.properties	18
3.3.3 CloudFrontSignedURLGenerator.properties	18
3.3.4 CloudFrontSignedCookieGenerator.properties	19
3.3.5 S3PreSignedURLGenerator.properties	21
3.4 Transcoder Configuration Files	21
3.4.1 Example Transcoder Configuration File	22
3.4.2 transcoder-service	23
3.5 Serving HLS Content	26
3.5.1 Using HLS Video Presets	26
3.5.2 Signed HLS Content	27

3.5.3	Serving HLS Content from CUE Front	28
3.6	Setting Section Parameters	31
3.7	Media Content Type Definition	31
3.7.1	Defining an AWS (Internal) Video Content Type	32
3.7.2	Defining a Brightcove Video Content Type	33
3.7.3	Defining an Internal Audio Content Type	34
3.7.4	Adding Pipeline and Preset Group Options	35
3.7.5	Including Key Frames in Video Content Items	36
3.8	Adding Watermarks To Video Content Items	38
3.8.1	Creating a Watermark Content Type	38
3.8.2	Configuring a Video Content Type to Use Watermarks	39
3.8.3	Uploading Watermark Images	40
3.8.4	Adding Watermark Section Parameters	40
3.8.5	Custom Watermarks	41
3.9	SSE Proxy Set-up	42
4	Automated Media Import	43
4.1	AWS	43
4.1.1	Enabling the AWS Media Import Service	43
4.1.2	AWS Media Import Service Configuration	43
4.1.3	AWSMediaImporterService.properties	43
4.1.4	Pub1ImporterConfiguration.properties	44
4.2	Brightcove	45
4.2.1	ExternalProviderPollerService.properties	45
4.2.2	BrightcoveProfile.properties	45
4.2.3	BcovMediaImporterConfig.properties	45
4.2.4	BrightcoveClientConfig.properties	46
5	Automated Media Publishing	47
5.1	Web Service-Based Automation	47
5.2	Syndication-Based Automation	48
5.3	Working With Video/Audio Fields	48
5.3.1	Specifying Crop Points	48
5.3.2	Specifying Closed Captions	49
5.3.3	Complete Video Field Example	49
6	Accessing Media from Templates	51

1 Introduction

The CUE Content Store's Video plug-in extends CUE with the functionality required to efficiently manage web site media content.

The Video plug-in provides:

- Fully-automated transcoding of video and audio content using Amazon Elastic Transcoder
- Fully-automated distribution of published video and audio content using Amazon CloudFront
- Fully-automated import of external videos from online video platforms such as [Brightcove](#)
- Simple Video and Audio editing functionality in CUE

The Video plug-in's objective is to make handling media content in the context of web site production as painless as possible, so the editing functionality added to CUE is a very simple **timeline editor** that is specifically geared to the needs of online publishing. The timeline editor allows you to:

- **Crop** an audio or video clip (that is select the segment you actually want to publish)
- Specify closed captions
- Select a **poster frame** for a video clip (a still that can be used to represent the clip when it is not playing)

All of these editing functions are **non-destructive**: that is, they do not modify the source media object, they only affect the published result.

The source media objects can either be **internal** or **external**, according to how they are stored:

- An **internal** media object is one that is stored under the control of the Content Store, in exactly the same way as an image or any other kind of binary file. A media object of this kind is usually acquired by uploading to CUE (although they can also be imported in various ways, just like any other content items). Although internal media objects are stored under the control of the Content Store, they are no longer stored locally: they are always stored in the cloud on Amazon S3. The terms **internal video/audio** and **AWS video/audio** are therefore interchangeable.
- An **external** media object is stored somewhere on the net - usually in a video cloud solution such as [Brightcove](#). The CUE content item doesn't contain the actual media object, it just contains a reference to its location on the net. Currently, Brightcove is the only external provider supported by the Video plug-in.

In order for the Video plug-in to work, the Content Store must use Amazon S3 as its back-end storage for binary files. For more about this, see [chapter 2](#).

1.1 About Amazon Elastic Transcoder

Amazon Elastic Transcoder is an Amazon web service that provides on-demand transcoding of media files between all commonly used audio and video formats. Once the Video plug-in has been correctly configured, it is largely invisible to plug-in users, simply providing a background service. Two Elastic

Transcoder concepts will, however, often be visible: when publishing a media file, the user will usually be able to choose between different **pipelines** and **preset groups**:

A **pipeline** is simply an Elastic Transcoder job queue. In a typical set-up there are two pipelines, one for normal jobs and one for high priority jobs, and the CUE user can choose which to use when publishing a media content item. There is no actual difference between the two pipelines other than their name, so the high priority pipeline will only work as a "fast lane" if it is actually reserved for a small number of high priority jobs. Pipelines are defined in Elastic Transcoder.

A **preset** is a set of parameters defining in great detail the output required from a transcoding job. For a video transcoding, it defines the file container format, video and audio codecs, bit rate, frame rate, width and height and so on. Presets are defined in Elastic Transcoder. A **preset group** is a named group of presets defined during the configuration of the Video plug-in. The idea is that a preset group should contain all the presets required to publish a particular type of media file in a particular context. Typically, a user will only need to choose between a small number of preset groups ("Standard Video" and "High Res Video", for example) and in some cases may not need to choose at all, since only one preset group has been defined.

The pipelines and preset groups available to the Video plug-in are defined in XML files called **transcoder configuration files**. Several of these files may be defined in order to provide different set-ups for different publications. For further information, see [section 3.4](#).

2 Installation

The following preconditions must be met before you can install Video 5.6.0-6 in Content Store:

- A suitable version of the CUE Content Store and CUE assembly tool have been installed as described in the **CUE Content Store Installation Guide** and are in working order.
- You have set up an Amazon S3 account that you can use for storing video files. For general information about how to do this, and how to configure the Content Store to use S3 for storage of binary files, see [Amazon S3 Storage](#). More specific instructions about the Video plug-in's use of S3 storage is provided in [chapter 3](#).
- You have set up an Amazon Elastic Transcoder account, and defined the **pipelines** and **presets** you are going to use. For details about how to do this, see [the Elastic Transcoder documentation](#)
- You have the credentials needed to access CCI Europe's SW repositories.

2.1 Conventions

The Content Engine and the software it depends on may be installed on one or several host machines depending on the type of installation required. In order to unambiguously identify the machines on which various installation actions must be carried out, the **CUE Content Store Installation Guide** defines a set of special host names that are used throughout the manual.

Some of these names are also used here:

assembly-host

The machine used to assemble the various Content Engine components into a enterprise archive or .EAR file.

engine-host

The machine(s) used to host application servers and Content Engine instances.

The host names always appear in a bold typeface. If you are installing everything on one host you can, of course, ignore them: you can just do everything on the same machine. If you are creating a larger multi-host installation, then they should help ensure that you do things in the right places.

In the following instructions, the following placeholders are used in some paths:

Placeholder	Path
<i>engine-installation</i>	/usr/share/escenic/escenic-content-engine-engine-version
<i>assemblytool_installation</i>	/usr/share/escenic/escenic-assemblytool

2.2 Installing

To install the Video on an Ubuntu or other Debian-based system, do the following on your **assembly-host** and on each of your **engine-hosts**:

1. Log in as **root**.
2. If necessary, add the CUE **apt** repository to your list of sources:

```
# echo "deb http://user:password@apt.escenic.com stable main non-free" >> /etc/
apt/sources.list.d/escenic.list
```

where *user* and *password* are your CUE download credentials (the same ones you use to access the CUE Maven repository). If you do not have any download credentials, please contact [CUE support](#).

3. Enter the following commands:

```
# apt-get update
# apt-get install escenic-video
```

On RedHat / CentOS systems, enter the following command as **root** on your **assembly-host** and each of your **engine-hosts**:

```
# rpm -Uvh https://user:password:yum.escenic.com/rpm/escenic-video-5.6.0-6.x86_64.rpm
```

where *version* is the correct version number of the package.

2.3 Re-assembling Applications

After installation, you **may** need to reassemble your web applications. You only need to reassemble if your installation includes any old-style JSP-based Escenic publications.

1. Log in as **escenic** on your **assembly-host**.
2. Run the **ece** script to re-assemble your Content Store applications

```
$ ece assemble
```

This generates an EAR file (**/var/cache/escenic/engine.ear**) that you can deploy on all your **engine-hosts**.

3. If you have a single-host installation, then skip this step.

On each **engine-host**, copy **/var/cache/escenic/engine.ear** from the **assembly-host**. If you have installed an SSH server on the **assembly-host** and SSH clients on your **engine-hosts**, then you can do this as follows:

```
$ scp escenic@assembly-host-ip-address:/var/cache/escenic/engine.ear /tmp/
```

where *assembly-host-ip-address* is the host name or IP address of your **assembly-host**.

4. On each **engine-host**, deploy the EAR file and restart the Content Store by entering:

```
$ ece stop deploy start --file /tmp/engine.ear
$ ece restart
```

2.4 Verifying The Installation

To verify the status of the Video plug-in, open the CUE Admin web application (usually located at <http://server/escenic-admin>) and click on **View installed plugins**. The status of all currently installed plug-ins is shown here, and indicated as follows:

The plug-in is correctly installed.

The plug-in is not correctly installed.

If the Video plug-in is correctly installed, you should see something like this in the displayed plug-in list:

2.5 Updating The Database Schema

The Video plug-in needs some additions to be made to the Content Store database schema. The scripts needed to make the required additions are included in the `misc/database/` folder of the distribution. There are two sets of scripts, one for MySQL databases, in `misc/database/mysql`, and one for Oracle databases in `misc/database/oracle`. There are three scripts in each folder:

- `constraints.sql`
- `indexes.sql`
- `tables.sql`

To run the scripts:

1. Log in as **escenic** on your **database-host**.
2. Copy or unpack the appropriate scripts for your database to an appropriate location (for example `/tmp/video/misc/database/mysql`).
3. Run the scripts as follows
 - For MySQL:

```
$ cd /tmp/video/misc/database/mysql/  
$ for e1 in tables.sql indexes.sql constraints.sql; do \  
mysql -u ece-user -pece-password -h dbhost db-name < $e1  
done;
```

replacing *db-name*, *dbhost*, *ece-user* and *ece-password* with the correct values for your database.

2.6 CUE Plug-in Activation

In order to be able to make use of Video plug-in functionality in CUE you need to activate CUE's Video plug-in. The plug-in is installed together with CUE itself, it just needs to be activated. To do this, you need to:

Video Plug-in User Guide

1. Log in as **root** on the host on which your CUE editor is installed.
2. Copy **/etc/escenic/cue-web-x.y.z/public/Video.yml** to a new file for editing.
3. Remove the **#** comment character from all lines in the copied file.
4. Save your changes.
5. Enter the following command to reconfigure CUE with the new settings:

```
| dpkg-reconfigure cue-web-x.y
```

3 Configuration

In order to be able to use the Video plug-in after installing it (see [chapter 2](#)) you must configure it correctly with the values needed to access and use various external services:

- Amazon Elastic Transcoder
- Amazon S3
- Amazon Cloudfront

The configuration settings need to be made in the following places:

- The Content Store's common configuration layer (see [Configuring The Content Engine](#) for general information about CUE configuration layers).
- Your publications' application configuration layers. An application configuration layer works in more or less the same way as other Content Store configuration layers, but it is publication-specific.
- One or more **transcoder configuration files**. These are XML files containing details of the Elastic Transcoder **pipelines** and **presets** to be used by your publications.
- Your publication's section parameters
- Your publications' **content-type** resource files (see [The Publication Resources](#) for general information about CUE resource files).

While you are working on the configuration, it is a good idea to set the Content Store logging level to **DEBUG** for the video plug-in. This will ensure that you get detailed information in the log file. Once the system is correctly configured you can set the logging level back to the default level. To change the logging level open a browser and go the **escenic-admin** application's **Logging Levels** page. Set the category **com.escenic.video** to the level you require (for example, **DEBUG**). For general information about setting logging levels, see [Logging](#).

3.1 Quick Start

You don't necessarily need to carry out **all** the configuration tasks described in this chapter in order to get the Video plug-in running. The following list describes the fastest route to a working media production line.

1. Copy example configuration files to your common configuration layer as described in [section 3.2](#).
2. Edit `/com/escenic/storage/filesystems/VideoFileSystemConfiguration.properties` as described in [section 3.2.1](#).
3. Edit `/com/escenic/storage/filesystems/KeyframeFileSystemConfiguration.properties` as described in [section 3.2.2](#).
4. Edit `/com/escenic/storage/Storage.properties` as described in [section 3.2.3](#).
5. Edit `/etc/escenic/engine/common/com/escenic/media/aws/DefaultTranscoderConfig.properties` as described in [section 3.2.4](#).
6. Edit `/etc/escenic/engine/common/com/escenic/media/services/AWSClient.properties` as described in [section 3.2.6](#).

7. If you will be producing HLS video (see [section 3.5.1](#)), then you may also want to edit `/etc/escenic/engine/common/com/escenic/media/services/AWSClientService.properties` as described in [section 3.2.7](#).
8. Edit `/etc/escenic/engine/common/com/escenic/media/aws/transcode-config.xml` as described in [section 3.4](#).
9. Edit `/etc/escenic/engine/common/com/escenic/media/aws/preview/PreviewTranscoderConfig.properties` as described in [section 3.2.8](#).
10. Add media content types to your publication content-type resources as described in [section 3.7.1](#) and [section 3.7.3](#). You only need to create content types for the media you are interested in (that is, if you are not interested in audio content, then you don't need to create an audio content type).
11. Configure your SSE Proxy as described in [section 3.9](#).

This quick-start configuration will give you an installation that:

- Uses the same transcoder configuration for all publications
- Uses S3 for storing transcoded media files, not Amazon CloudFront

If this does not meet all your requirements, then you will need to edit some additional configuration files, as described in the following sections.

3.2 Common Configuration Layer Set-up

This set-up task consists of copying example configuration files from the plug-in installation into the Content Store common configuration layer (if necessary) and then modifying the copied files to meet your requirements. In detail, you must:

1. Log in to your Content Store host as the `escenic` user.
2. If you installed the Video plug-in using the old-style installation method, copy all the supplied common configuration layer files as follows:

```
$ cp -r engine-installation/plugins/video-5.6.0-6/misc/siteconfig/common/com/escenic \  
> /etc/escenic/engine/common/com
```

If you installed the Video plug-in using the new package-based installation method, then this step is not required: the configuration files are automatically installed in the correct location.

3. Edit the copied files as described in the following sections.

3.2.1 VideoFileSystemConfiguration.properties

The full path of this file is `/com/escenic/storage/filesystems/VideoFileSystemConfiguration.properties`. This file is used to define the cloud storage location in which you will store your video files.

For a detailed description of how to configure cloud storage file system components in general, see [Create FileSystemConfiguration Components](#).

3.2.2 KeyframeFileSystemConfiguration.properties

The full path of this file is `/com/escenic/storage/filesystems/KeyframeFileSystemConfiguration.properties`. It is used to define the cloud storage location in which Amazon Elastic Transcoder stores the key frame files it generates during transcoding.

The key differences between this file and `/com/escenic/storage/filesystems/VideoFileSystemConfiguration.properties` are that:

- The `baseURI` property must be set to point to the Amazon bucket that you set up as the thumbnail output bucket when configuring Amazon Elastic Transcoder
- It has the following additional property setting:

```
readOnly=true
```

which tells the Content Store that it may not use this S3 location for storing files, only for reading them.

For a detailed description of how to configure cloud storage file system components in general, see [Create FileSystemConfiguration Components](#).

3.2.3 Storage.properties

The full path of this file is `/com/escenic/storage/Storage.properties`. If you don't already have a copy of this file in your common configuration layer, then you will need to create it. It determines which storage systems the Content Store uses for different kinds of files. It needs to reference the `VideoFileSystemConfiguration` and `KeyframeFileSystemConfiguration` components you have defined (plus possibly other cloud storage locations that you may have defined previously).

For a detailed description of how to configure this file in general, see [Create FileSystemConfiguration Components](#).

When you are setting the `fileSystemConfigurations` property in this file, you must be sure to use **absolute paths** to specify the location of the `VideoFileSystemConfiguration` and `KeyframeFileSystemConfiguration` components (and any other file system components that will be used by the Video plug-in, should there be any). For example:

```
fileSystemConfigurations=/com/escenic/storage/filesystems/  
KeyframeFileSystemConfiguration,/com/escenic/storage/filesystems/  
VideoFileSystemConfiguration
```

This is only required for file system components used by the Video plug-in, it is not a general requirement.

3.2.4 DefaultTranscoderConfig.properties

The full path of this file is `/etc/escenic/engine/common/com/escenic/media/aws/DefaultTranscoderConfig.properties`. You need to set the following property in this file:

serviceDefinition

The path to the XML configuration file containing your transcoding service configuration. For example:

```
serviceDefinition=transcode-config.xml
```

For further information, see [section 3.4](#).

If you want to have different transcoding set-ups for different publications, then you will need to create several copies of this file with different names (for example **TranscoderConfigPub1.properties**, **TranscoderConfigPub2.properties** and so on, and set the **serviceDefinition** property to reference a different transcoder configuration file in each copy. For example:

```
serviceDefinition=pub1-transcode-config.xml
```

For each additional copy of this file that you create, you also need to add an entry referencing the file to **/etc/escenic/engine/common/com/escenic/media/aws/TranscodingConfigFactory.properties** (see [section 3.2.5](#)).

3.2.5 TranscodingConfigFactory.properties

The full path of this file is **/etc/escenic/engine/common/com/escenic/media/aws/TranscodingConfigFactory.properties**.

If you only want one transcoding set-up for all your publications, then you do not need to edit this file.

If you want to have different transcoding set-ups for different publications, then you need to set the following properties:

defaultTranscodingConfig

This property must be set to reference your default transcoder configuration, for example:

```
defaultTranscodingConfig=./DefaultTranscoderConfig
```

publicationTranscodingConfigs.pub-name

If you have made copies of **DefaultTranscoderConfig.properties** containing different transcoding configurations for different publications, then you can specify several **publicationTranscodingConfigs** properties to reference them, where *pub-name* is the name of a publication that requires specialized transcoding.

The value of each property must be the name of a configuration file containing a specialized transcoding configuration, for example:

```
publicationTranscodingConfigs.DailyNews=./TranscoderConfigDailyNews
publicationTranscodingConfigs.WeeklyGossip=./TranscoderConfigEntertainment
publicationTranscodingConfigs.Entertainment=./TranscoderConfigEntertainment
```

Note that, as in the example above, you can share a configuration file between several publications.

You do not need to specify an entry for all your publications. If a publication is not specified here, then the default transcoding configuration specified with the **defaultTranscodingConfig** property is used.

3.2.6 AWSClientConfig.properties

The full path of this file is **/etc/escenic/engine/common/com/escenic/media/services/AWSClientConfig.properties**. You need to set the following properties in this file:

accessKey

An access key for accessing your Amazon Web Services (AWS) account. This is the access key you got when you set up your Amazon S3 storage (see [Amazon S3 Storage](#)). For information about AWS access keys and AWS identity management in general, see the [AWS IAM documentation](#).

```
| accessKey=aws-access-key
```

secretKey

The secret key for the access key specified with the **accessKey** property. This is the access key you got when you set up your Amazon S3 storage (see [Amazon S3 Storage](#)).

```
| secretKey=aws-secret-key
```

region

The name of the AWS Elastic Transcoder region you want to use. For example:

```
| region=US_EAST_1
```

For a list of the currently available Elastic Transcoder regions, see [Amazon Elastic Transcoder](#).

protocol

The protocol to use when generating pre-signed URLs for accessing content stored in S3. The allowed values are **HTTP** (the default) and **HTTPS**. For example:

```
| protocol=HTTP
```

3.2.7 AWSClientService.properties

The full path of this file is `/etc/escenic/engine/common/com/escenic/media/services/AWSClientService.properties`. If you are using Elastic Transcoder to produce HLS video streams, you can control the length of the `.ts` segments in the streams by setting the following property:

segmentDuration

The duration of each `.ts` segment in generated HLS streams, specified in seconds. Allowed values are in the range 1 to 60. For example:

```
| segmentDuration=15
```

The default value is 10.

For more information about HLS video, see [section 3.5](#).

3.2.8 PreviewTranscoderConfig.properties

The full path of this file is `/etc/escenic/engine/common/com/escenic/media/aws/preview/PreviewTranscoderConfig.properties`.

In order for the CUE media player / timeline editor to work it requires low resolution versions (often called **proxy versions**) of media objects to work with. As soon as a media object is uploaded or imported into a media content item, therefore, the Video plug-in sends it to Amazon Elastic Transcoder to be transcoded to the required proxy format. **PreviewTranscoderConfig.properties** contains the configuration settings required to access the Elastic Transcoder for this purpose.

If you do not define a **PreviewTranscoderConfig.properties** file, then the CUE media player / timeline editor will not work. You will be able to publish media content, but you will not be able to preview or edit it in CUE.

You need to set the following properties in this file:

pipelineID (required)

The ID of the Elastic Transcoder pipeline to which proxy version transcoding jobs are to be submitted. For example:

```
| pipelineID=pipeline-id
```

audioPresetID (required)

The ID of the Elastic Transcoder preset you have created for generating audio proxy versions. For example:

```
| audioPresetID=preset-id
```

videoPresetID (required)

The ID of the Elastic Transcoder preset you have created for generating video proxy versions. For example:

```
| videoPresetID=preset-id
```

3.2.9 Poller.properties

The full path of this file is **/etc/escenic/engine/common/com/escenic/media/status/Poller.properties**.

The settings in this file control the service that submits media content for transcoding.

You can **optionally** set the following properties in this file:

enableService

Determines whether or not the service is enabled. It is enabled by default on all hosts, although it only ever actually runs on one host. You should never disable it in the common configuration layer, although you can disable it on specific hosts (presentation hosts, for example) by copying the file to a host-specific configuration layer and setting the property to false.

pollingInterval

Specifies how often the service should check for media content that needs transcoding, in milliseconds. The default value of 30000 means that the service checks for new or modified media content every 30 seconds, and if it finds any, submits it to Elastic Transcoder for transcoding. You might want to shorten this interval, for example:

```
| pollingInterval=10000
```

3.2.10 StateChangeNotifier.properties

The full path of this file is **/etc/escenic/engine/common/com/escenic/media/services/StateChangeNotifier.properties**.

You can **optionally** use it to enable notifications of the following events related to media content items:

- Transcoding completed

- Content item published
- Content item unpublished

Notifications are sent to the author and creator of a content item and to the initiators of any of the above events.

In order for notifications to work, the Content Store [Notes plug-in](#) must be installed.

You only need to set one property in this file to enable notifications:

```
serviceEnabled=true
```

3.3 Application Configuration Layer Set-up

This set-up task consists of copying example configuration files from the plug-in installation into your publications' application configuration layers and then modifying the copied files to meet your requirements. In detail, you must:

1. Log in to your Content Store host as the **escenic** user.
2. For each publication, create an application configuration layer (if it does not already exist) as follows:

```
$ mkdir -p /etc/escenic/engine/webapp/pub-name/com/escenic
```

where *pub-name* is the name of the publication.

3. Copy all the supplied application configuration layer files as follows:

```
$g cp -r engine-installation/plugins/video-5.6.0-6/misc/siteconfig/webapp/com/escenic/video \> /etc/escenic/engine/webapp/pub-name/com/escenic
```

4. Edit the copied files as described in the following sections.

3.3.1 AWSMediaInfoGenerator.properties

This **.properties** file is only used for CUE Front-based publications. For old JSP-based publications you should use **VideoArticleDecorator.properties** instead (see [section 3.3.2](#)).

The full path of this file is:

```
/etc/escenic/engine/webapp/video-presentation-webservice/com/escenic/media/aw/AWSMediaInfoGenerator.properties
```

This file is used to configure the **video-presentation-webservice** webapp. This webapp provides a webservice at **/mediaInfo** that supplies information about the different variants available for published videos. The properties you can set are:

defaultURLGenerator

How you set this property depends on whether or not you are using CloudFront to distribute your transcoded media content, and whether or not you want to make use of CloudFront signed cookies:

- If you are using CloudFront, then you should either set it to:

```
defaultURLGenerator=/com/escenic/video/presentation/CloudFrontURLGenerator
```

or, if you want to use signed cookies:

```
defaultURLGenerator=/com/escenic/video/presentation/  
CloudFrontSignedCookieGenerator
```

- If you are not using CloudFront then you should set it to:

```
defaultURLGenerator=/com/escenic/video/presentation/S3PreSignedURLGenerator
```

This is the default setting if you do not specify **defaultURLGenerator**.

URLGeneratorProfiles

The **defaultURLGenerator** property sets the default behavior of the **video-presentation-webservice** for all video content. You may, however, want the web service to handle content differently depending on which publication it belongs to – if, for example, only some publications' video content is distributed by CloudFront. If this is the case, then you need to create one or more URL generator profiles (see [section 3.3.1.1](#)), and add references to them in this property. For example:

```
URLGeneratorProfiles=./profiles/URLGeneratorProfilePub1.properties,./profiles/  
URLGeneratorProfilePub1.properties
```

3.3.1.1 URL Generator Profiles

A URL generator profile is a **.properties** file with the following content:

```
$class=com.escenic.media.profile.URLGeneratorProfile  
  
publicationName=publication-name  
URLGenerator=url-generator-path
```

where:

- *publication-name* is the name of a publication
- *url-generator-path* is the path to the URL generator that is to be used for this publication - that is, one of the following:

```
/com/escenic/video/presentation/CloudFrontURLGenerator  
/com/escenic/video/presentation/CloudFrontSignedCookieGenerator  
/com/escenic/video/presentation/S3PreSignedURLGenerator
```

For example:

```
$class=com.escenic.media.profile.URLGeneratorProfile  
  
publicationName=pub1  
URLGenerator=/com/escenic/video/presentation/S3PreSignedURLGenerator
```

You only need to create URL generator profiles if you want to use different URL generators for different publications. Any publication for which you do not define a URL generator will make use of the default URL generator specified in **AWSMediaInfoGenerator.properties** (see [section 3.3.1](#)).

You can call your URL generator profiles anything you like, and store them wherever you like. It's a good idea, however, to store them together with `AWSMediaInfoGenerator.properties` (in a `profiles` subfolder, for example).

3.3.2 VideoArticleDecorator.properties

This `.properties` file is only required to support old JSP template-based publications. For CUE Front-based applications, you should use `AWSMediaInfoGenerator.properties` instead (see [section 3.3.1](#)).

The full path of this file is:

```
/etc/escenic/engine/webapp/pub-name/com/escenic/video/presentation/  
VideoArticleDecorator.properties
```

where `pub-name` is the name of the publication. You only need to set the following property in this file:

URLGenerator

How you set this property depends on whether or not you are using CloudFront to distribute your transcoded media content. If you are using CloudFront, then you should set it as follows:

```
URLGenerator=./CloudFrontSignedURLGenerator
```

If you are not using CloudFront then you should set it as follows:

```
URLGenerator=./S3PreSignedURLGenerator
```

Any other properties defined in the file should be left unmodified.

3.3.3 CloudFrontSignedURLGenerator.properties

The full path of this file is:

```
/etc/escenic/engine/webapp/pub-name/com/escenic/video/presentation/  
CloudFrontSignedURLGenerator.properties
```

where `pub-name` is the name of the publication.

You only need to edit this file if you are using CloudFront to distribute your transcoded media content. If you are not using CloudFront, then you should edit `S3PreSignedURLGenerator.properties` instead (see [section 3.3.5](#)). Set the following properties:

httpprotocol

The protocol you want to be used to access your distributed media files. For example:

```
protocol=http
```

The allowed values are:

- `http`
- `https`
- `rtmp` (for Flash streaming)

distributionDomain

The domain name at which the media files are to be published. CloudFront generates a domain name when you create a distribution, consisting of a long id followed by ".cloudfront.net". You can, however, configure CloudFront to use a domain name of your own, such as:

```
distributionDomain=media.mycompany.com
```

For more information about CloudFront distribution domain names and how to replace them with your own, see [Getting Started with CloudFront](#).

privateKeyFile

The path of a file containing a CloudFront private key. For example:

```
privateKeyFile=/etc/escenic/engine/webapp/demo/com/escenic/video/presentation/cloudfront-pk.der
```

CloudFront generates public/private key pairs that you can use to provide access to restricted content (paid content, for example). The keys are used to create **signed URLs**: automatically generated URLs that allow a customer access to restricted content for a limited period. For more information about this, see [Serving Private Content through CloudFront](#).

A private key should under no circumstances be shared with any third party, including CCI Europe and Amazon.

keyPairId

The ID of the CloudFront public/private key used to secure your CloudFront content. This is supplied to you along with the private key at the time it is generated.

```
keyPairId=key-pair-id
```

expiredTime

The lifetime of the pre-signed URLs generated by CloudFront, specified in minutes. Once this period is over, a pre-signed URL will no longer work and the user will no longer have access to the restricted content. The default value is 900 (= 15 hours).

```
expiredTime=900
```

3.3.4 CloudFrontSignedCookieGenerator.properties

The full path of this file is:

```
/etc/escenic/engine/webapp/video-presentation-webservice/com/escenic/video/presentation/CloudFrontSignedCookieGenerator.properties
```

You only need to edit this file if you are using CloudFront to distribute your transcoded media content **and** you want to use signed cookies rather than signed URLs for access control. The use of signed cookies protects the **.m3u8** manifests and **.ts** video fragments in HLS video streams as well as ordinary media files, ensuring that they can only be accessed via your site. Set the following properties:

httpprotocol

The protocol you want to be used to access your distributed media files. For example:

```
protocol=http
```

The allowed values are:

- **http**
- **https**

distributionDomain

The domain name at which the media files are to be published. CloudFront generates a domain name when you create a distribution, consisting of a long id followed by ".cloudfront.net". You must, however, configure CloudFront to use your own domain name which you must also specify here. The domain name must be identical to or a subdomain of the domain name from which the videos are played. If, for example, the videos are played from **mysite.com**, then this property must also be set to **mysite.com** or a subdomain of **mysite.com**, for example:

```
| distributionDomain=media.mysite.com
```

For more information about CloudFront distribution domain names and how to replace them with your own, see [Getting Started with CloudFront](#).

privateKeyFile

The path of a file containing a CloudFront private key. For example:

```
| privateKeyFile=/etc/escenic/engine/webapp/demo/com/escenic/video/presentation/  
cloudfront-pk.der
```

CloudFront generates public/private key pairs that you can use to provide access to restricted content (paid content, for example). The keys are used to create **signed** cookies: automatically generated cookies that allow a customer access to restricted content for a limited period. For more information about this, see [Using Signed Cookies](#). For detailed information about how to set up public/private key pairs on CloudFront, see [here](#). Note in particular that the private key file must be DER encoded and must have the extension **.der**.

A private key should under no circumstances be shared with any third party, including CCI Europe and Amazon.

keyPairId

The ID of the CloudFront public/private key used to secure your CloudFront content. This is supplied to you along with the private key at the time it is generated.

```
| keyPairId=key-pair-id
```

expiredTime

The lifetime of the signed cookies generated by CloudFront, specified in minutes. Once this period is over, a signed cookie will no longer work and the user will no longer have access to the restricted content. The default value is **900** (= 15 hours).

```
| expiredTime=900
```

cookieDomain

The domain of the host to which the cookie should be sent. If your domain is **myservice.com**, for example, and your CloudFront domain is **cloud.myservice.com**, then you should explicitly set this property to **myservice.com**. If your domain and CloudFront domain are identical then you can omit this property, since the default is the host of the current document URL (excluding any subdomains).

cookieAttributes

Any other cookie attributes you want to set, separated by semicolons. For example:

```
| cookieAttributes=Secure;HttpOnly
```

For descriptions of all available cookie attributes list, see [here](#). You are recommended to always include the **HttpOnly** attribute, for security reasons.

3.3.5 S3PreSignedURLGenerator.properties

The full path of this file is:

```
/etc/escenic/engine/webapp/pub-name/com/escenic/video/presentation/
S3PreSignedURLGenerator.properties
```

where *pub-name* is the name of the publication.

You only need to edit this file if you are **not** using CloudFront to distribute your transcoded media content. If you **are** using CloudFront, then you should edit `CloudFrontSignedURLGenerator.properties` instead (see [section 3.3.3](#)).

expiredTime

All content stored in S3 buckets is private. In order to allow users to access content, pre-signed URLs are generated. These URLs have a limited lifetime: once one expires, the content it referenced is no longer accessible. This property specifies the lifetime of the pre-signed URLs, in minutes. The maximum allowed value is **10080** (= 7 days). For further information, see [Amazon S3 Authentication Policy](#). The default value is **900** (= 15 hours).

```
| expiredTime=900
```

3.4 Transcoder Configuration Files

A transcoder configuration file is an XML file containing details of the Amazon Elastic Transcoder **pipelines** and **presets** available for use with a particular publication or set of publications. A pipeline is a queue for holding transcoding jobs, and a preset is a set of parameters defining how to transcode between two specified formats. Neither pipelines nor presets are **defined** in a transcoder configuration file: they must be defined using Elastic Transcoder, and are then simply referenced from transcoder configuration files. For further information, see [the Elastic Transcoder documentation](#).

You should have already defined the pipelines and presets you are going to need in Elastic Transcoder. You use the transcoder configuration files to specify which pipelines and presets are going to be used by your publications. If all your publications are going to use the same pipelines and presets, then you only need one transcoder configuration file, and you can use the default `/etc/escenic/engine/common/com/escenic/media/aws/transcode-config.xml`:

1. Log in to your Content Store host as the **escenic** user.
2. Open `/etc/escenic/engine/common/com/escenic/media/aws/transcode-config.xml` for editing.
3. Edit the file to make use of the Elastic Transcoder pipelines and presets you have defined. For further information see [section 3.4.1](#) and [section 3.4.2](#).

If, on the other hand, your publications have different transcoding requirements, you may need to create several different files. Copy the supplied `transcode-config.xml` as many times as required, for example:

1. Log in to your Content Store host as the **escenic** user.
2. Copy the supplied `transcode-config.xml` as many times as required, for example:

```
| $ cd /etc/escenic/engine/common/com/escenic/media/aws/transcode-config.xml
| $ cp transcode-config.xml transcode-config-dailynews.xml
```

```
$ cp transcode-config.xml transcode-config-entertainment.xml
```

3. Edit all the files to make use of the Elastic Transcoder pipelines and presets you have defined. For further information see [section 3.4.1](#) and [section 3.4.2](#).
4. Make sure that the files you create are correctly referenced by the transcoder configurations in your common configuration (see [section 3.2.4](#)). Check the `serviceDefinition` property in your `DefaultTranscoderConfig.properties` file and make sure it correctly references your default transcoder configuration file (`transcode-config.xml`). Similarly, check any copies that you have made of `DefaultTranscoderConfig.properties` and make sure that their `serviceDefinition` properties point to the correct transcoder configuration files (`transcode-config-dailynews.xml` or `transcode-config-entertainment.xml`, for example).

3.4.1 Example Transcoder Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<service-definition xmlns="http://xmlns.escenic.com/2012/transcoder-service"
  xmlns:ui="http://xmlns.escenic.com/2008/interface-hints">
  <pipelines>
    <pipeline default="true" name="low-priority">
      <ui:title>Normal</ui:title>
      <id>pipeline-id</id>
    </pipeline>
    <pipeline name="high-priority">
      <ui:title>High priority</ui:title>
      <id>pipeline-id</id>
    </pipeline>
  </pipelines>
  <presetGroups>
    <presetGroup type="video" name="low-res">
      <ui:title>Low resolution videos</ui:title>
      <presets>
        <preset id="preset-id">
          <thumbnails/>
        </preset>
        <preset id="preset-id"/>
        <preset id="preset-id"/>
      </presets>
    <presetGroup type="video" name="high-res" default="true">
      <ui:title>High resolution videos</ui:title>
      <presets>
        <preset id="preset-id">
          <thumbnails/>
        </preset>
        <preset id="preset-id"/>
        <preset id="preset-id"/>
        <preset id="hls-preset-id"/>
        <preset id="hls-preset-id"/>
        <preset id="hls-preset-id"/>
      </presets>
    </presetGroup>
  </presetGroups>
</service-definition>
```

This example defines a service that provides two pipelines and two preset groups. *pipeline-id* and *preset-id* in a real service definition file would be replaced by Elastic Transcoder IDs - long, unique strings of characters generated when you create a pipeline or preset. Note the following:

- Two pipelines, one of which is for high priority jobs, is a typical set-up. There is no actual difference between the two pipelines. The idea is that most jobs are put in the "normal" pipeline, which may result in long queues during busy periods. Since the "high priority" pipeline is reserved for occasional high priority jobs, long queues should not arise. The prioritization mechanism is therefore dependent on disciplined use.
- A **preset group** is a group of Elastic transcoder presets suitable for a particular purpose: in the example above, **low-res** contains the presets needed to convert low-resolution videos to all the required output formats. **high-res** does the same for high resolution videos, and includes a set of **HLS** presets that are combined to form an HLS stream. See [section 3.5.1](#) for more about this. What preset groups you create is entirely determined by your particular requirements.
- A preset group must contain either only audio presets or only video presets, and must have a **type** attribute indicating what kind of preset group it is.
- One preset group of each type (audio and video) must have the attribute **default="true"** set.
- The **ui:title** elements are used to specify the names displayed in CUE pipeline and preset group selection controls.

3.4.2 transcoder-service

The **transcoder-service** schema defines the Escenic Video plug-in transcoder service definition format. You can use it to define the Amazon Elastic transcoder pipelines and presets available for use in a specific publication or group of publications.

Namespace URI

The namespace URI of the **transcoder-service** schema is **http://xmlns.escenic.com/2012/transcoder-service**.

Root Element

The root of a **transcoder-service** file must be a **service-definition** element.

3.4.2.1 id

The pipeline ID. This is a number generated by Amazon Elastic Transcoder when a pipeline is created.

Syntax

```
<id>
  text
</id>
```

3.4.2.2 pipeline

Represents an Elastic Transcoder transcoder **pipeline**. A pipeline is a queue for transcoding jobs. For more information see [the Elastic Transcoder documentation](#).

Syntax

```
<pipeline
  default="(true|false)"?
  name="NCName"
  >
  <ui:title>...</ui:title>
```

```
<id>...</id>
</pipeline>
```

Attributes

default="(true|false)" (optional)

Whether or not this is the default **pipeline**. One **pipeline** element should have **default** set to **true**, and one only.

name="NCName"

The name of the **pipeline** element.

3.4.2.3 pipelines

Contains definitions of the pipelines available in this transcoder service.

Syntax

```
<pipelines>
  <pipeline>...</pipeline>+
</pipelines>
```

3.4.2.4 preset

Represents an Elastic Transcoder **preset**. A preset is a template for transcoding media files from one format to another. For more information see [the Elastic Transcoder documentation](#).

Syntax

```
<preset
  id="text"
/>
```

Attributes

id="text"

The preset ID. This is a number generated by Amazon Elastic Transcoder when a preset is created.

3.4.2.5 presetGroup

Defines a preset group. A preset group is a group of **presets**.

Syntax

```
<presetGroup
  type="(audio|video)"
  name="NCName"
  default="(true|false)"?
  >
  <presets>...</presets>
</presetGroup>
```

Attributes

type="(audio|video)"

The media type this **presetGroup** is intended to handle.

Allowed values are:

audio

The **presetGroup** is intended to handle audio content.

video

The **presetGroup** is intended to handle video content.

name="NCName"

The name of the **presetGroup** element.

default="(true|false)" (optional)

Whether or not this is the default **presetGroup**. One **presetGroup** element should have **default** set to **true** for each setting of the **type** attribute. That is, there should be one default preset group for video, and one default preset group for audio.

3.4.2.6 presetGroups

Contains definitions of the preset groups available in this transcoder service.

Syntax

```
<presetGroups>
  <presetGroup>...</presetGroup>+
</presetGroups>
```

3.4.2.7 presets

A container for **preset** elements.

Syntax

```
<presets>
  <preset/>+
</presets>
```

3.4.2.8 service-definition

The root element of an Escenic Video plug-in transcoder service definition.

Syntax

```
<service-definition>
  <pipelines>...</pipelines>
  <presetGroups>...</presetGroups>
</service-definition>
```

3.4.2.9 ui:title

The display name of a pipeline or preset group. It is used when displaying pipeline/preset group options in Content Studio.

Syntax

```
<ui:title>
  text
</ui:title>
```

3.5 Serving HLS Content

[HLS](#) is a standard for **adaptive streaming**, which allows clients to adjust the bitrate of a video stream to match the currently available bandwidth. The way HLS works is that the server produces several streams of different quality (i.e, different bitrates), each of which is divided up into short, fixed-duration segments (say 10 seconds each). The client is then able to monitor the available bandwidth, and if necessary switch stream every 10 seconds in order to maintain the best possible video stream.

This section contains information about various things you need to take account of when serving HLS content.

3.5.1 Using HLS Video Presets

In order to provide an HLS stream, you have to use HLS presets. Amazon Elastic Transcoder provides a range of predefined HLS presets, or you can define your own. An Elastic transcoder HLS preset generates one of the substreams that go to make up a complete HLS stream. So if you define a preset group like this:

```
<presetGroup type="video" name="high-res" default="true">
  <ui:title>High resolution videos</ui:title>
  <presets>
    <preset id="preset-id">
      <thumbnails/>
    </preset>
    <preset id="preset-id"/>
    <preset id="preset-id"/>
    <preset id="hls-preset-id"/>
    <preset id="hls-preset-id"/>
    <preset id="hls-preset-id"/>
  </presets>
</presetGroup>
```

Then the result is four transcoded outputs: three fixed bit-rate videos and one HLS video stream offering three different bitrates.

The short video segments that make up an HLS video stream are stored in files with the extension **.ts**. Each substream is represented by a playlist file with the extension **.m3u8**. This is a simple text file that contains correctly ordered references to all the **.ts** files in the stream. The whole HLS stream is represented by yet another **master playlist .m3u8** file that contains references to each of the substreams. This structure is automatically created by the Elastic Transcoder, and is the structure expected by HLS-capable clients. All you need to be aware of is that the HLS presets you define in a preset group get merged to a single HLS output stream, represented by a **.m3u8** playlist file.

You can control the length of the **.ts** segments generated by Elastic Transcoder by setting a **segmentDuration** property (see [section 3.2.7](#)). This will determine how quickly a client responds to changes in available bandwidth.

3.5.2 Signed HLS Content

The standard "signed URL" access control method used by the Video plug-in works fine for downloadable media content, but does not fully prevent unauthorized access to HLS content. Signed URLs prevent unauthorized access to top level media files, but do not protect the underlying `.m3u8` files and the `.ts` segment files that hold the actual video content. Amazon CloudFront offers the use of a signed cookies method as an alternative to signed URLs, which solves this problem. Using signed cookies, all the components of an HLS stream are only accessible via your web site, and cannot be directly accessed by users. This method also protects downloadable media content, so it is a complete replacement for the use of signed URLs that also fully protects HLS content.

The signed cookies feature is only available on Amazon CloudFront. It is not available when serving media content directly from S3. It is also only available for CUE Front-based publications – you cannot use this feature with JSP-based publications.

To make use of the signed cookies feature you need to:

- Create a `CloudFrontSignedCookieGenerator.properties` file (see [section 3.3.4](#))
- Set the `defaultURLGenerator` property in `MediaInfoGenerator.properties` to `/com/escenic/video/presentation/CloudFrontSignedCookieGenerator` (see [section 3.3.1](#)). Alternatively, you may want to set up a publication-specific profile containing this setting (see [section 3.3.1.1](#)).
- Make sure that both S3 and CloudFront are set up with the correct CORS policies:
 - For a site with the domain name `mysite.com`, the S3 output bucket would need a CORs configuration like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>http://mysite.com</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <ExposeHeader>ETag</ExposeHeader>
    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
</CORSConfiguration>
```

If content is being served via CUE front, then your Cook endpoints must also be included as allowed origins. For the `tomorrow-online` publication, for example, you would need to specify:

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>http://mysite.com</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <ExposeHeader>ETag</ExposeHeader>
    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
  <CORSRule>
    <AllowedOrigin>http://tomorrow-online.mysite.com</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <ExposeHeader>ETag</ExposeHeader>
    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
</CORSConfiguration>
```

```

    <AllowedOrigin>http://tomorrow-online.mysite.com:8101</AllowedOrigin> <!--
port 8101 used for cookies -->
    <AllowedMethod>GET</AllowedMethod>
    <ExposeHeader>ETag</ExposeHeader>
    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
</CORSConfiguration>

```

- Set up CloudFront to forward the following headers:

```

Access-Control-Allow-Credentials
Access-Control-Allow-Origin
Access-Control-Request-Headers
Access-Control-Request-Method
Origin

```

For more information about Cloudfront cache configuration and CORs, see [here](#) and [here](#).

3.5.3 Serving HLS Content from CUE Front

The **tomorrow-online** demo publication supplied with CUE Front handles video content in general, but does not support HLS streaming. You can modify it to support HLS streaming using the [VideoJS](#) video player as follows:

1. Make sure the videoJS library is loaded by adding the following lines to **templates/_patterns/01-globals/05-nonvisual/_scriptsBlock.twig**:

```

<script src="https://vjs.zencdn.net/7.8.4/video.js"></script>
<script src="https://unpkg.com/@videojs/http-streaming@2.1.0/dist/videojs-http-streaming.min.js"></script>

```

2. Include the VideoJS stylesheet in **templates/_patterns/01-globals/05-nonvisual/_stylesheet.twig**:

```

<link href="https://vjs.zencdn.net/7.8.4/video-js.css" rel="stylesheet" />

```

3. Make the following highlighted changes to **templates/_patterns/02-atoms/03-videos/04-cloud-video.twig**:

```

{% set video = article.mediaInfo.videos[1] %} {# Change the index value to the
appropriate video source #}
{% set captions = article.caption_href %}
<div class="video-teaser">
  {% if article.mediaInfo.headers is not empty %}
    {% for header in article.mediaInfo.headers if header.key == "Set-Cookie" %}
      {{ set_header("Set-Cookie", header.value) }}
    {% endfor %}
  {% endif %}
  <video id="video-js-{{ article.id }}"
    class="video-js"
    width="{{ video.width }}"
    height="{{ video.height }}"
    controls>
    {# <source src="{{ video.uri }}" type="{{ video.mimeType }}">#}
    {% if captions is not empty %}
      <track kind="subtitles" src="{{ captions | relativize }}" default>
    {% endif %}
    Your browser does not support HTML5 video.
  </video>
  <script type="text/javascript">

```

```

document.addEventListener("DOMContentLoaded", () => {
  const player = videojs("video-js-{{ article.id }}");
  player.pause();
  player.fluid(true);
  player.src({
    src: '{{ video.uri }}',
    type: '{{ video.mimeType }}',
    withCredentials: true,
  });
  player.load();
});
</script>
</div>

```

4. To be able to render inline HLS video in storylines, you also need to make the following highlighted additions to `templates/_patterns/02-atoms/11-storyline-elements/storyline-element-video.twig`:

```

{% set video = element.relation.mediaInfo.videos[1] %}
{% set autoplay = 0 %}
{% set allowFullScreen = 0 %}
{% set closedcaption = 0 %}

{% for fieldValue in element.fields %}
  {% if fieldValue.name == 'closedcaption' and fieldValue.booleanValue == 1 %}
    {% set closecaption = 1 %}
  {% endif %}
  {% if fieldValue.name == 'autoplay' and fieldValue.booleanValue == 1 %}
    {% set autoplay = 1 %}
  {% endif %}
  {% if fieldValue.name == 'allowfullscreen' and fieldValue.booleanValue == 1 %}
    {% set allowFullScreen = 1 %}
  {% endif %}
{% endfor %}

{% if element.relation.mediaInfo.headers is not empty %}
  {% for header in element.relation.mediaInfo.headers if header.key == "Set-Cookie" %}
    {{ set_header("Set-Cookie", header.value) }}
  {% endfor %}
{% endif %}

<video id="video-js-{{ element.relation.id }}"
  class="video-js mb2"
  width="{{ video.width }}"
  height="{{ video.height }}"
  controls>
  {# <source src="{{element.relation.mediaInfo.videos[1].uri}}" >#}
  <track label="English"
    srclang="en"
    src="{{ element.relation.caption_href }}"
    {% if closecaption == 1 %}
      kind="subtitles"
    {% endif %}
    default
  >
  Your browser does not support HTML5 video.
</video>

```

```

{% if element.relation.fields.description is not empty %}
  <div class="f6 header-font mt3 primary mb4">
    {{ element.relation.fields.description }}
  </div>
{% endif %}

<script type="text/javascript">
  document.addEventListener("DOMContentLoaded", () => {
    const player = videojs("video-js-{{ element.relation.id }}", {
      autoplay: {% if autoplay == 1 %}true{% else %}false{% endif %},
      BigPlayButton: true,
      fluid: true,
      children: {
        controlBar: {
          subtitlesButton: true,
          captionsButton: true,
          fullscreenToggle: {% if allowFullScreen == 1 %} true {% else %} false {%
endif %}
        }
      }
    });

    player.pause();
    player.src({
      src: '{{ video.uri }}',
      type: '{{ video.mimeType }}',
      withCredentials: true,
    });
    player.load();

    player.ready(() => {
      {% if allowFullScreen == 0 %}
      // Disable entering full screen on double click
      player.tech_.off('dblclick');
      {% endif %}
    });
  });
</script>

```

5. If you are using signed cookies for access control, then you will also need to add the following GraphQL code to `recipe/queries/video.graphql`:

```

... on Video {
  mediaInfo {
    headers {
      key
      value
    }
  }
}

```

This makes sure that the requires headers are included in the JSON data returned by the Cook. The template changes listed above for `04-cloud-video.twig` and `03-videos/04-cloud-video.twig` include code for forwarding any supplied headers:

```

{% if article.mediaInfo.headers is not empty %}
  {% for header in article.mediaInfo.headers if header.key == "Set-Cookie" %}
    {{ set_header("Set-Cookie", header.value) }}
  {% endfor %}

```

```
| {% endif %}
```

If you are **not** intending to use signed cookies, then you can omit this code from both templates.

3.6 Setting Section Parameters

You can override the default global or publication level **pipeline** or **presetGroup** settings for part of a publication by setting section parameters in one of the publication's section. The section parameters to use are:

video.transcoder.config.pipeline

If specified, this parameter sets the default pipeline to be used for transcoding video content in the current section and all its subsections. The default can be overridden for individual content items if video content types are configured with a pipeline option (see [section 3.7.4](#)).

audio.transcoder.config.pipeline

If specified, this parameter sets the default pipeline to be used for transcoding audio content in the current section and all its subsections. The default can be overridden for individual content items if audio content types are configured with a pipeline option (see [section 3.7.4](#)).

video.transcoder.config.presetGroup

If specified, this parameter sets the default preset group to be used for transcoding video content in the current section and all its subsections. The default can be overridden for individual content items if video content types are configured with a preset group option (see [section 3.7.4](#)).

audio.transcoder.config.presetGroup

If specified, this parameter sets the default preset group to be used for transcoding audio content in the current section and all its subsections. The default can be overridden for individual content items if audio content types are configured with a preset group option (see [section 3.7.4](#)).

3.7 Media Content Type Definition

In order to be able to use the Video plug-in, you need to add at least one suitably configured media content type to your publication's **content-type** resource. For general information about the **content-type** resource and how to edit it, see the [CUE Content Store Resource Reference](#).

A media content type must either be a **video** content type or an **audio** content type (not currently supported), and both of them can either be **internal** or **external**:

Internal

An internal media content type includes a **link** field for holding an actual video/audio object (an MPEG or MP3 file, for example). Such content items can therefore hold video/audio content, not just a reference to the content. An internal content item is usually created in CUE by uploading the video/audio content that is to be published (although it can also be imported in various ways, just like any other content). It is actually stored in the Amazon cloud and is therefore also often referred to as an AWS video.

External

An external media content type doesn't contain the actual video, it just contains a reference to content stored in some other location - usually a video cloud solution such as [Brightcove](#). Currently, Brightcove is the only external provider supported by the Video plug-in. The plugin must be configured with Brightcove credentials in order to be able to automatically import videos from Brightcove.

The Video Plug-in distribution file contains an example **content-type** resource file (**video/misc/example/content-type.xml**) with more complete examples of the content type definitions described in the following sections.

3.7.1 Defining an AWS (Internal) Video Content Type

The AWS video content type must at least have the following:

- A **media** child element belonging to the namespace **http://xmlns.escenic.com/2013/media**. This element must have a **type** attribute set to **video**, thereby identifying the content type as a video content type managed by the Video plug-in.
- A parameter called **com.escenic.article.staging** set to **false**, which specifies that [content item staging](#) must be disabled for this content type. If content item staging is disabled generally at your installation or for the whole publication, then you can omit this parameter.
- A **link field** for holding a link to a locally stored video file. The **link** element must have a child **relation** element containing the value **com.escenic.edit-media**.
- A **basic** field with:
 - **mime-type** set to **application/json**
 - a **video** sub-element with an **enabled** attribute set to **true**. This element must belong to the namespace **http://xmlns.escenic.com/2010/video**.
- A decorator called **videoArticleDecorator**
- An interface-hints element called **list-style** set to **video** to show proper listing at CUE.

The following example shows such a minimal **content-type**:

```
<content-type name="aws-video">
  <media xmlns="http://xmlns.escenic.com/2013/media" type="video" enabled="true"
  dropable="true"/>
  <parameter name="com.escenic.article.staging" value="false"/>
  <ui:decorator name="videoArticleDecorator"/>
  <panel name="main">
    <field name="binary" type="link">
      <relation>com.escenic.edit-media</relation>
    </field>
    <field name="video" type="basic" mime-type="application/json">
      <video xmlns="http://xmlns.escenic.com/2010/video" enabled="true"/>
    </field>
  </panel>
</content-type>
```

When a new content item of this type is created in CUE, the is required to upload a suitable file to the **link** field. You can use a **constraints** element to limit the file types it is possible to upload. For example:

```
<field name="binary" type="link">
```



```
<constraints>
  <mime-type>video/mpeg</mime-type>
  <mime-type>video/mp4</mime-type>
  <mime-type>application/mxf</mime-type>
</constraints>
</field>
```

Here is a more realistic version of an AWS video content type, this time including a title field, label and so on, but with the important parts highlighted.

```
<content-type name="aws-video">
  <media:media type="video"/>
  <parameter name="com.escenic.article.staging" value="false"/>
  <ui:icon>graphic</ui:icon>
  <ui:label>VideoCloud video</ui:label>
  <ui:title-field>title</ui:title-field>
  <ui:decorator name="videoArticleDecorator"/>
  <ui:list-style>video</ui:list-style>
  <panel name="main">
    <field name="title" type="basic" mime-type="text/plain"/>
    <field name="binary" type="link">
      <constraints>
        <mime-type>video/mpeg</mime-type>
        <mime-type>video/mp4</mime-type>
        <mime-type>application/mxf</mime-type>
      </constraints>
    </field>
    <field name="video" type="basic" mime-type="application/json">
      <video xmlns="http://xmlns.escenic.com/2010/video" enabled="true"/>
    </field>
  </panel>
</content-type>
```

3.7.2 Defining a Brightcove Video Content Type

The Brightcove video content type must at least have the following:

- A **media** child element belonging to the namespace `http://xmlns.escenic.com/2013/media`. This element must have an **enabled** attribute set to **true**.
- A parameter called `com.escenic.article.staging` set to **false**, which specifies that [content item staging](#) must be disabled for this content type. If content item staging is disabled generally at your installation or for the whole publication, then you can omit this parameter.
- A **basic** video field with:
 - **mime-type** set to `application/json`
 - a **video** sub-element belonging to the namespace `http://xmlns.escenic.com/2010/video` with an **enabled** attribute set to **true**.
- A decorator called `bcovVideoArticleDecorator`
- An interface-hints element called **list-style** set to **video** to show proper listing at CUE.

- A **basic** external media metadata field with:
 - **mime-type** set to **application/json**
 - a **media** sub-element belonging to the namespace **http://xmlns.esenic.com/2013/media**.

The **external-media-metadata** field should be a hidden field. It contains the actual Brightcove video JSON data. You can retrieve any information you need about the video from this field. You can, for example, retrieve tags that have been added to the video in Brightcove.

Here is an example Brightcove video content type with the important parts highlighted:

```
<content-type name="brightcove-video">
  <media:external-video enabled="true"/>
  <parameter name="com.esenic.article.staging" value="false"/>
  <ui:icon>graphic</ui:icon>
  <ui:label>Brightcove Video</ui:label>
  <ui:title-field>title</ui:title-field>
  <ui:decorator name="bcovVideoArticleDecorator"/>
  <ui:list-style>video</ui:list-style>
  <panel name="main">
    <field name="title" type="basic" mime-type="text/plain"/>
    <field name="video" type="basic" mime-type="application/json">
      <video xmlns="http://xmlns.esenic.com/2010/video" enabled="true"/>
    </field>
    <field name="external-media-metadata" type="basic" mime-type="application/json">
      <media:external-media/>
      <ui:label>External media JSON</ui:label>
      <ui:hidden/>
    </field>
  </panel>
</content-type>
```

3.7.3 Defining an Internal Audio Content Type

A internal audio content type must at least have the following:

- A **media** child element belonging to the namespace **http://xmlns.esenic.com/2013/media**. This element must have a **type** attribute set to **audio**, thereby identifying the content type as an audio content type managed by the Video plug-in.
- A parameter called **com.esenic.article.staging** set to **false**, which specifies that [content item staging](#) must be disabled for this content type. If content item staging is disabled generally at your installation or for the whole publication, then you can omit this parameter.
- A **link field** for holding a link to a locally stored audio file. The **link** element must have a child **relation** element containing the value **com.esenic.edit-media**.
- A **basic** field with:
 - **mime-type** set to **application/json**
 - an **audio** child element with an **enabled** attribute set to **true**. This element must belong to the namespace **http://xmlns.esenic.com/2013/audio**.
 - a **ui:hidden** element which prevents it from being displayed in CUE.
- A decorator called **audioArticleDecorator**

The following example shows such a minimal **content-type**:

```
<content-type name="internal-audio">
  <media xmlns="http://xmlns.escenic.com/2013/media" type="audio"/>
  <parameter name="com.escenic.article.staging" value="false"/>
  <ui:icon>graphic</ui:icon>
  <ui:label>Internal audio</ui:label>
  <ui:title-field>title</ui:title-field>
  <ui:decorator name="audioArticleDecorator"/>
  <panel name="main">
    <field name="title" type="basic" mime-type="text/plain"/>
    <field name="binary" type="link">
      <relation>com.escenic.edit-media</relation>
    </field>
    <field name="audio" type="basic" mime-type="application/json">
      <audio xmlns="http://xmlns.escenic.com/2013/audio" enabled="true"/>
    </field>
  </panel>
</content-type>
```

When a new content item of this type is created in CUE, the user is required to upload a suitable file to the **link** field. You can use a **constraints** element to limit the file types it is possible to upload. For example:

```
<field name="binary" type="link">
  <constraints>
    <mime-type>audio/mp3</mime-type>
    <mime-type>audio/mp4</mime-type>
  </constraints>
</field>
```

A real internal audio content type will of course usually contain other fields and elements too - see the expanded example in [section 3.7.1](#).

3.7.4 Adding Pipeline and Preset Group Options

Adding pipeline and preset group options to media content types enable editorial users to override default pipeline and preset group settings for individual content items.

A pipeline/preset group selection option is a **field** element that:

- Has **mime-type** set to **text/plain**
- Has a child **pipeline** or **preset-group** element that belongs to the namespace **http://xmlns.escenic.com/2013/media** and has the attribute **enabled = "true"**

A minimal pipeline option field definition, in other words, would look like this:

```
<field name="pipeline" type="basic" mime-type="text/plain">
  <ui:label>Pipeline</ui:label>
  <pipeline xmlns="http://xmlns.escenic.com/2013/media" enabled="true"/>
</field>
```

In practice, however, such a field definition would require CUE users to know the names of the available pipelines and enter them correctly. The preferred method is to use a **collection field**. A collection field can be configured to get all currently defined pipeline names from an Atom feed provided by the Video plug-in. The retrieved names are then offered as suggestions when the user starts typing. Here is an example of the recommended way to define a pipeline option field:

```
<field name="pipeline" type="collection" mime-type="text/plain"
  src="escenic/video/config/pipelines" select="content">
  <ui:label>Pipeline</ui:label>
  <pipeline xmlns="http://xmlns.escenic.com/2013/media" enabled="true"/>
</field>
```

The **src** attribute contains the path of the Video plug-in's pipeline Atom feed and must be set to **escenic/video/config/pipelines**. The **select** attribute must be set to **content**.

A preset group option field can be created in exactly the same way:

```
<field name="preset" type="collection" mime-type="text/plain"
  src="escenic/video/config/presets/video" select="content">
  <ui:label>Preset Group</ui:label>
  <preset-group xmlns="http://xmlns.escenic.com/2013/media" enabled="true"/>
</field>
```

For video preset groups the **field** element's **src** attribute must be set to **escenic/video/config/presets/video**, while for audio preset groups, it must be set to **escenic/video/config/presets/audio**:

```
<field name="preset" type="collection" mime-type="text/plain"
  src="escenic/video/config/presets/audio" select="content">
  <ui:label>Preset Group</ui:label>
  <preset-group xmlns="http://xmlns.escenic.com/2013/media" enabled="true"/>
</field>
```

For a detailed description of how collection fields work, see [Collection Fields](#).

3.7.5 Including Key Frames in Video Content Items

Key frames are still images generated from a video clip. They can be hand-selected images of significant points in the video, or they can be automatically generated at fixed intervals. Key frames can be used to represent the video in various contexts - a keyframe image is usually used, for example, to represent the video when it is not being played.

Amazon Elastic Transcoder can be configured to generate key frames (called **thumbnails** in the Elastic Transcoder user interface) at fixed intervals. If you want to save the keyframes and associate them with the video to which they belong, then you need to add the following items to your **content-type** resource:

A content type for holding the key frames

This content type needs to contain a **link** field for storing the key frame images. A minimal key frame content type might look like this:

```
<content-type name="keyframe">
  <ui:label>Key Frame</ui:label>
  <ui:title-field>name</ui:title-field>
  <panel name="main">
    <ui:label>Image content</ui:label>
    <field name="name" type="basic" mime-type="text/plain">
      <ui:label>Name</ui:label>
      <constraints>
        <required>true</required>
      </constraints>
    </field>
    <field type="link" name="binary">
```

```

        <relation>com.escenic.edit-media</relation>
        <constraints>
            <mime-type>image/jpeg</mime-type>
            <mime-type>image/png</mime-type>
        </constraints>
    </field>
</panel>
</content-type>

```

A relation definition

This relation definition is needed to associate key frame content items with their source video content items. It might look like this:

```

<relation-type-group name="default-relation-type-group">
    <relation-type name="keyframes">
        <ui:label>Key frame images</ui:label>
    </relation-type>
</relation-type-group>

```

Relation references

You will need to add a reference to the above relation definition to each of the video content types that you want to include key frames. For example:

```

<content-type name="aws-video">
    <media xmlns="http://xmlns.escenic.com/2013/media" type="video" enabled="true"
dropable="true"/>
    <parameter name="com.escenic.article.staging" value="false"/>
    <ui:decorator name="videoArticleDecorator"/>
    <panel name="main">
        <field name="binary" type="link">
            <relation>com.escenic.edit-media</relation>
        </field>
        <field name="video" type="basic" mime-type="application/json">
            <video xmlns="http://xmlns.escenic.com/2010/video" enabled="true"/>
        </field>
        <ref-relation-type-group name="default-relation-type-group"/>
    </panel>
</content-type>

```

A store-keyframes element

This element causes the Video plug-in to store the generated key frames as content items of the correct type. The element must belong to the namespace `http://xmlns.escenic.com/2010/video` and must have the following attributes:

- **content-type**, specifying the name of the key frame content type you have defined
- **relation**, specifying the name of the key frame relation you have defined
- **state(optional)**, specifying the state to be applied to generated key frame content items. If not specified, the default state is **published**
- **relate-keyframe-state(optional)**, specifying the state in which generate key frame content items will be related. It's value only be **submitted** or **published**. If not specified, the default state is **published**

You must add such an element to the video field in each of the video content types that you want to include key frames. For example:

```

<content-type name="aws-video">
    <media xmlns="http://xmlns.escenic.com/2013/media" type="video" enabled="true"
dropable="true"/>

```

```
<parameter name="com.escenic.article.staging" value="false"/>
<ui:decorator name="videoArticleDecorator"/>
<panel name="main">
  <field name="binary" type="link">
    <relation>com.escenic.edit-media</relation>
  </field>
  <field name="video" type="basic" mime-type="application/json">
    <video xmlns="http://xmlns.escenic.com/2010/video" enabled="true"/>
      <store-keyframes xmlns="http://xmlns.escenic.com/2010/video"
        content-type="keyframe" relation="keyframes"
        state="draft"/>
    </field>
  <ref-relation-type-group name="default-relation-type-group"/>
</panel>
</content-type>
```

For information about configuring Amazon Elastic Transcoder to actually generate the key frames (or thumbnails) you want, see [Thumbnail Settings](#).

3.8 Adding Watermarks To Video Content Items

A watermark is an image (such as a television channel logo) that is overlaid on a broadcast video. The Amazon Elastic Transcoder supports the addition of PNG and JPEG watermarks to videos during transcoding. Up to four different watermarks can be overlaid on a video in this way. The characteristics of these watermarks (size, position and so on) must be defined in your Elastic Transcoder presets.

Before you can configure the Video plug-in to add watermarks to your videos, therefore, you need to make sure that your Elastic Transcoder presets contain the watermark definitions you will be using. Each watermark definition has an ID that you will need to use when you are configuring the Video plug-in. CCI Europe recommends that you use the following watermark IDs in all your presets: **TopLeft**, **TopRight**, **BottomLeft** and **BottomRight**.

For general information about Elastic Transcoder watermarks, see [AWS Watermark documentation](#). For information about how to create presets, see [Creating a Preset in Elastic Transcoder](#).

Once you have created the Elastic Transcoder presets, you can configure a publication to add watermarks to videos by:

- Creating a content type for storing watermark images
- Configuring your video content type(s) by adding a **watermarks** element
- Optionally extend the video content type(s) with one or more watermark relations in order to enable detailed control over the watermarking of individual videos
- Uploading the watermark images you want to use
- Adding watermark section parameters to one or more of the sections in your publication

These steps are described in the following sections.

3.8.1 Creating a Watermark Content Type

A watermark content type is a standard image content type with a **link** field for storing a reference to the watermark image. In addition, however, it must have a **parameter** that identifies it as a

watermark image. This parameter must have the name `com.escenic.video.watermark.image` and the value `true`.

Here is a simple example of a watermark content type:

```
<content-type name="watermark">
  <parameter name="com.escenic.video.watermark.image" value="true"/>
  <ui:label>Watermark</ui:label>
  <ui:title-field>name</ui:title-field>
  <panel name="main">
    <ui:label>Image content</ui:label>
    <field name="name" type="basic" mime-type="text/plain">
      <ui:label>Name</ui:label>
      <constraints>
        <required>true</required>
      </constraints>
    </field>
    <field type="link" name="binary">
      <relation>com.escenic.edit-media</relation>
      <constraints>
        <mime-type>image/jpeg</mime-type>
        <mime-type>image/png</mime-type>
      </constraints>
    </field>
  </panel>
</content-type>
```

3.8.2 Configuring a Video Content Type to Use Watermarks

In order to configure a video content type to make use of watermarks, you need to add a **watermarks** element to the content type. This element ensures that the Video plug-in stores the watermarks correctly. The **watermarks** element must belong to the namespace `http://xmlns.escenic.com/2010/video`. It has two attributes:

content-type (required)

This attribute specifies the name of the watermark image content type you have defined.

disable-field (optional)

This attribute specifies the name of a boolean field in this content type that can be used to disable watermarking for the current article. You must then also add such a field to the content type. If you add such a field, then any video content item in which it is set to true will not be watermarked, even if it belongs to a section in which watermarking is enabled.

The following example shows a video content type that includes a "Disable Watermarking" field:

```
<content-type name="aws-video">
  <media xmlns="http://xmlns.escenic.com/2013/media" type="video" enabled="true"
  dropable="true"/>
  <parameter name="com.escenic.article.staging" value="false"/>
  <ui:decorator name="videoArticleDecorator"/>
  <panel name="main">
    <field name="binary" type="link">
      <relation>com.escenic.edit-media</relation>
    </field>
    <field name="video" type="basic" mime-type="application/json">
      <video xmlns="http://xmlns.escenic.com/2010/video" enabled="true"/>
      <watermarks xmlns="http://xmlns.escenic.com/2010/video">
```

```

        content-type="watermark" disable-field="disable-watermark" />
    </field>
    <ref-relation-type-group name="default-relation-type-group"/>
</panel>
</content-type>

```

You can also optionally add watermark relations to the content type in order to enable custom watermarking of individual videos. For more about this, see [section 3.8.5](#).

3.8.3 Uploading Watermark Images

Once you have made the required configuration changes to your publication's **content-type** resource, and uploaded it to the Content Store, you can upload the watermark images you want to use. Start Content Store and create a watermark content item for each watermark. You will be prompted to supply an image file to upload. Note that the image file you supply **must** have one of the following file extensions:

- .jpg
- .jpeg
- .png

The file extension must be lower case as shown above. Amazon Elastic Transcoder uses the file extension to determine the file type, and only recognises these three extensions. A file with a different extension will be rejected even if it is in fact of the correct type.

Once you have created each content item, make a note of its ID, as you will need to use it when adding section parameters to the publication.

3.8.4 Adding Watermark Section Parameters

The final step is to add some section parameters to your publication in order to control which watermark images are actually added to videos in various contexts. CUE section parameters are inheritable, so if you want to use the same watermarks everywhere in your publication, then you can just set these parameters in your publication's root section. If you want different watermarks in a particular section, then you can override the global settings by redefining the parameters in that section.

You need to set two different parameters:

com.escenic.aws.watermark.content_ids

This parameter specifies which watermarks are to be added to videos. It has the following form:

```
com.escenic.aws.watermark.content_ids = watermark-image-id-list
```

where *watermark-image-id-list* is a comma-separated list of content item IDs identifying the watermark images to be used. For example:

```
com.escenic.aws.watermark.content_ids=12345,23456
```

com.escenic.aws.watermark_id.content-id

One parameter like this must be added for each content item ID specified with **com.escenic.aws.watermark.content_ids**. Each parameter has the form:

```
com.escenic.aws.watermark_id.content-id=watermark-id
```

where:

- *content-id* is a watermark image content ID.
- *watermark-id* is an Elastic Transcoder watermark ID that effectively determines where the watermark will be placed on the video.

For example:

```
com.escenic.aws.watermark_id.12345=TopRight
com.escenic.aws.watermark_id.23456=BottomLeft
```

If you want to switch watermarking off in a section of your publication, you can do so by setting **com.escenic.aws.watermark.content_ids** to no value in that section. For example:

```
com.escenic.aws.watermark.content_ids =
```

For general information about section parameters and how to set them, see http://docs.escenic.com/ece-pub-admin-guide/7.15/the_edit_section_parameters_page.html.

Note that the watermarking rules you define with these section parameters can be overridden for individual videos if you enable custom watermarking as described in [section 3.8.5](#).

3.8.5 Custom Watermarks

The basic configuration described in the preceding sections provides section-level watermark control. That is, the details of which watermarks to use and where they are to appear in the video frame is determined at section level (often, in fact, in the root section, so that all videos in a publication get the same watermarks). It is not possible for different videos in the same section to have different watermarks. You can allow users to completely disable watermarking on specific videos by setting the **disable-field** attribute, but that is all.

It is, however, possible to enable control at the individual video level by adding a set of relations to the video content type. You add one relation type for each watermark position that you want to use. For the top left watermark position, for example, you need to add something like this to your video content type:

```
<relation-type name="Top-Left" max="1">
  <watermarks id="TopLeft" xmlns="http://xmlns.escenic.com/2010/video"/>
  <ui:label>Top Left Watermark</ui:label>
  <allow-content-types>
    <ref-content-type name="watermark" />
  </allow-content-types>
</relation-type>
```

Make sure that:

- The **relation-type** element has a **max** constraint attribute set to 1.
- The **watermarks** element belongs to the **http://xmlns.escenic.com/2010/video** namespace.
- The **watermarks** element's **id** attribute matches the watermark ID used for this position in your section parameters.
- The **allow-content-types** constraint is correctly set to only allow watermark content items in the relation.

With watermark relations defined in this way, CUE users can control the watermarks on individual videos. Section-level watermark configurations then effectively function as default settings. When a video content item is first saved in CUE, the watermarks specified at section level are added to the video and appear in the watermark relations that have been defined. The user can then change the default settings by dragging watermark content items into or out of the watermark relations.

Note that you cannot use custom watermarks in some positions and fixed watermarks in others. If, for example, you add a watermark relation in the top left position as in the example shown above, then this will become the only position in which watermarks can be displayed. Setting a top right watermark at the section level, for example, will not work. If you want to be able to display watermarks in the top right corner, you must add a top right watermark relation as well. So you have to choose between a basic configuration providing fixed watermarks in all defined positions, or a custom watermarks configuration providing custom watermarks in all defined positions.

To change the watermark of a published video in CUE, the video must first be set back into a draft state. The watermark can then be updated and the video republished.

3.9 SSE Proxy Set-up

You need to add a setting to your Content Store's SSE Proxy. The Content Store uses SSE (Server-sent Events) to send information about content changes to CUE clients, so that CUE does not need to poll the Content Store for changes. The Content Store sends these SSE notifications via an **SSE Proxy**, a lightweight web proxy installed somewhere in the network. For general information about the SSE Proxy and how it is installed and Configured, see [SSE Proxy](#).

In order to be able to work with video or audio content in CUE, you need to add an extra back-end service definition to the SSE Proxy's `sse-proxy.yaml` configuration file. The file will already contain a back-end service definition for basic Content Store events that should look something like this:

```
backends:
  - uri: "http://editorial.example.com:8081/webservice/escenic/changelog/sse"
    credentials:
      username: "someuser"
      password: "verysecret"
```

All you need to do is add a second service definition with the URI `http://engine-host:port/webservice/escenic/video/mediaStatusSSE`. `engine-host` and `port` should be the same as for the existing entry, and so should the user name and password. For example:

```
backends:
  - uri: "http://editorial.example.com:8081/webservice/escenic/changelog/sse"
    credentials:
      username: "someuser"
      password: "verysecret"
  - uri: "http://editorial.example.com:8081/webservice/escenic/video/mediaStatusSSE"
    credentials:
      username: "someuser"
      password: "verysecret"
```

4 Automated Media Import

This section contains detailed information about how to import media content automatically from different cloud service providers such as AWS and Brightcove. The Video plugin includes import services that can be configured to automatically run a number of independent import tasks.

4.1 AWS

The import service for AWS periodically checks a specified folder path on Amazon S3 storage for new media content to import. If files are available, they are imported to a specified destination folder (also on Amazon S3 storage) as defined in configuration files. How many different import tasks you define is up to you. You will need to create at least one task for each publication to which you want to import media content.

4.1.1 Enabling the AWS Media Import Service

In order to enable the import service, you need to open the file `configuration-root/Initial.properties` for editing in one of your configuration layers, and add the following line (at the bottom of the file):

```
service.3.3-aws-import-service=/com/escenic/media/aws/services/AWSMediaImporterService
```

4.1.2 AWS Media Import Service Configuration

You need to:

1. For each publication, create a common configuration layer (if it does not already exists) as follows:

```
$ mkdir -p /etc/escenic/engine/common/com/escenic/media/aws/services/import/publ
```

2. To import AWS media, copy all the supplied common configuration layer files to the required locations as follows:

```
$ cp -r engine-installation/plugins/video-5.6.0-6/misc/siteconfig/common/com/escenic/media/aws/services/AWSMediaImporterService.properties \
> /etc/escenic/engine/common/com/escenic/media/aws/services
$ cp -r engine-installation/plugins/video-5.6.0-6/misc/siteconfig/common/com/escenic/media/services/MediaImporterConfiguration1.properties \
> /etc/escenic/engine/common/com/escenic/media/aws/services/import/publ/PublImportConfiguration.properties
```

If you installed the Video plug-in using the new package-based installation method, then this step is not required: the configuration file are automatically installed in the correct location.

3. Edit the configuration files as described in the following sections.

4.1.3 AWSMediaImporterService.properties

The full path of this file is:

```
/etc/escenic/engine/common/com/escenic/media/aws/services/  
AWSMediaImporterService.properties
```

importerConfigurations

Set this property as follows:

```
importerConfigurations=./import/pub1/Pub1ImportConfiguration
```

You can if necessary specify multiple configuration files separated by commas:

```
importerConfigurations=./import/pub1/Pub1ImportConfiguration,./import/pub2/  
Pub2ImportConfiguration,./import/pub3/Pub3ImportConfiguration
```

4.1.4 Pub1ImporterConfiguration.properties

The full path of this file is:

```
/etc/escenic/engine/common/com/escenic/media/aws/services/import/pub1/  
Pub1ImporterConfiguration.properties
```

publicationId

The ID of the publication in which the content item is to be created. For example:

```
publicationId=1
```

sectionId

The ID of the section in which the content item is to be created. For example:

```
sectionId=8
```

state

The state to which the created content item is to be set: **draft**, **submitted** or **published**. For example:

```
state=draft
```

contentType

The type of the created content item. For example:

```
contentType=aws-video
```

syndicationBucketBaseURI

The base URI of the AWS syndication bucket – the root folder to which media files are to be uploaded. For example:

```
syndicationBucketBaseURI=s3://bucket-name/storage/path/
```

errorBucketBaseURI

The base URI of the AWS error bucket – the root folder to which failed media files will be copied. It must have a different path than **syndicationBucketBaseURI**. For example:

```
errorBucketBaseURI=s3://bucket-name/storage/path/
```

mediaType

The media type of the imported content: **Video**, **Audio** or **Image**. This property is optional – if you don't specify a value then the media type is detected. If, however, you know that only one specific media type is to be imported, then you are recommended to set it. For example:

```
mediaType=Video
```

4.2 Brightcove

The Brightcove import service periodically checks the Brightcove video cloud for new videos to import to Escenic. This service only imports videos that have already been transcoded in Brightcove.

Assuming you already have copied all configuration files to your common configuration layer as described in [section 3.2](#), then all you need to do to configure the Brightcove import service is edit the configuration files described in the following sections.

4.2.1 ExternalProviderPollerService.properties

The full path of this file is `/etc/escenic/engine/common/com/escenic/media/ext/poll/ExternalProviderPollerService.properties`. You need to set the following properties in this file.

serviceEnabled

It controls the external media import service, which is disabled by default. In order to enable the service, add the following property.

```
| serviceEnabled=true
```

interval

This property determines how frequently the import service runs. By default, the interval between the termination of one execution and the commencement of the next is 30000 milliseconds. You can, however, override it as follows:

```
| interval=20000
```

4.2.2 BrightcoveProfile.properties

The full path of this file is `/etc/escenic/engine/common/com/escenic/media/ext/BrightcoveProfile.properties`. It contains the following property:

```
| startDate=start-date
```

which you can use to control how far back in time the import service will look for videos to import. By default, the import service uses the current date as start date, which means that old videos in the Brightcove video cloud from before you started using the import service will not be imported. If you want (for example) to ensure that videos from the last year (for example) are imported, then you should set the **startDate** property to the required date.

The required date format is `yyyy-MM-dd hh:mm:ss`. The specified date is compared with the Brightcove cloud videos' "last modified" dates. So if you specify:

```
| startDate=2016-01-01 00:00:00
```

then all Brightcove videos created or modified since the beginning of 2016 will be imported.

4.2.3 BcovMediaImporterConfig.properties

The full path of this file is `/etc/escenic/engine/common/com/escenic/media/bcov/BcovMediaImporterConfig.properties`. You need to set the following properties in this file:

publicationId

The id of the publication in which media content items are to be created.

```
| publicationId=publication-id
```

sectionId

The id of the section in which media content items are to be created. By default, it takes the default section of the publication. However, you can override this.

```
| sectionId=section-id
```

state

The state to be assigned to newly-created media content items: **draft**, **submitted** or **published**. By default, the content items are assigned the state **draft**. You can, however, override this. For example:

```
| state=submitted
```

contentType

The content type of the media content items created by the import service. For example:

```
| contentType=external-media
```

4.2.4 BrightcoveClientConfig.properties

The full path of this file is `/etc/escenic/engine/common/com/escenic/media/ext/client/BrightcoveClientConfig.properties`. You need to set the following properties in this file

accountId

Your [Brightcove](#) video cloud account id

```
| accountId=accound-id
```

clientId

The client ID used to access Brightcove. For information on how to get a client ID, see [Managing API Authentication Credentials](#).

```
| clientId=client-id
```

clientSecret

The secret needed to authenticate your Brightcove client ID. For details, see [Client credential flow](#).

```
| clientSecret=client-secret
```

5 Automated Media Publishing

This section contains detailed information about how to implement automated media publishing workflows. There are two basic methods you can use to implement such a workflow:

- Web service-based automation, where media content is passed to the Content Store via its REST web service
- Syndication-based automation, where media content is passed to the Content Store via its import/export or **syndication** subsystem

Whichever method you use, the basic objective is the same. You need to:

1. Define a content item of the correct type in the Content Store.
2. Create the defined content item.
3. Change the status of the content item to **published**. This causes the Content Store to send the video/audio to Amazon Elastic Transcoder for transcoding and key frame generation.

5.1 Web Service-Based Automation

The recommended way to implement an automated media publishing process is to use the Content Store's web service. This web service allows you to interact with the Content Store by sending HTTP requests to the web service. For a description of the web service and how to use it in general, see the [CUE Content Store Integration Guide](#). The web service allows an external process to perform most of the operations that end users can perform from CUE. In this case the objective is to create a video/audio content item, which involves the following steps:

1. POST the media file you want to publish to the Content Store web service. The web service has a special fixed URI to which media files and other binary files can be POSTed
2. GET the web service URI for the publication/section to which the content item is to be added.
3. Make an XML document (specifically, an Atom entry) containing all the information required to create the required content item and POST it to the correct URI.

The process of creating a media content item is in fact no different from the process of creating other kinds of binary content items (images, PDFs, office documents and so on). It is described in detail in the following sections of the **CUE Content Store Integration Guide**:

- [Navigate The Section Hierarchy](#) tells you how to get the upload URL of the section you want to add the content item to.
- [Create a Content Item](#) tells you how to create and upload a content item in general.
- [Creating Binary Content items](#) provides the additional information you need about uploading binary files to the Content Store web service.
- [Creating Content Items in Different States](#) tells you how to set the state of the content item you create. In order to trigger transcoding and publishing, you need to set the state of the content item to **published**.

5.2 Syndication-Based Automation

You can also implement an automated video/audio publishing process using the Content Store's syndication format. This is a proprietary CCI Europe XML file format that can be used for import/export purposes. It is described in detail in the [CUE Content Store Syndication Reference](#).

To define and create a content item in this way you simply create a syndication file containing all the required information and upload it to a specified import folder on the Content Store server. The Content Store will then automatically import the file and create a corresponding content item, triggering the transcoding and key frame generation process.

For a general description of how to use the Content Store's import service, see [The Import Service](#).

Note also that, in order to trigger transcoding, the content items must be published on import, by setting the `content` element's `state` attribute to `published`.

5.3 Working With Video/Audio Fields

A video content item must have a **video field** to store various items of video-specific information. The video field usually has the name "video", but if it doesn't you can find out which is the content item's video field by examining the `content-type` resource. The video field is the one that contains a `video` element, like this:

```
<field name="video" type="basic" mime-type="application/json">
  <video xmlns="http://xmlns.escenic.com/2010/video" enabled="true"/>
</field>
```

The video field can be used to specify JSON data defining crop points and/or closed captions, as described in the following sections.

5.3.1 Specifying Crop Points

You can specify that the loaded video or audio clip is to be cropped before it is transcoded. To do this you must include a `playback` key-value pair in the JSON data. For example:

```
{
  ...
  "playback":{
    "in":1.5,
    "out":5.6
  }
  ...
}
```

The value of the `playback` key must be a map containing the following key-value pairs:

in

The point at which the transcoded, published clip is to start, measured from the start in seconds.

out

The point at which the transcoded, published clip is to end, measured from the start in seconds.

5.3.2 Specifying Closed Captions

You can include a **timeline** containing closed captions for the video in the JSON data:

```
{
  ...
  "timeline": {
    "tracks": {
      "OverlayDefault": [
        {
          "startTime": 2.5,
          "id": "780d02d5-6ba3-46f1-ab82-ffc8e8f09522",
          "endTime": 4.0,
          "title": "Hello"
        },
        {
          "startTime": 5.0,
          "id": "24f2ba48-7741-4053-bf3a-d4a3e3409bb1",
          "endTime": 7.5,
          "title": "world"
        }
      ]
    }
  }
  ...
}
```

OverlayDefault is an array containing a sequence of captions. Each caption definition must include the following fields:

startTime

The point at which display of the caption is to start, measured from the start in seconds.

id

An ID for the caption.

title

The caption text.

endTime

The point at which display of the caption is to end, measured from the start in seconds.

5.3.3 Complete Video Field Example

Here is a complete JSON example showing the content of a video field:

```
{
  "timeline": {
    "tracks": {
      "OverlayDefault": [
        {
          "startTime": 2.5,
          "id": "780d02d5-6ba3-46f1-ab82-ffc8e8f09522",
          "endTime": 4.0,
          "title": "Hello"
        },
        {
          "startTime": 5.0,
          "id": "24f2ba48-7741-4053-bf3a-d4a3e3409bb1",
          "endTime": 7.5,

```

```
        "title": "world"  
      }  
    ]  
  }  
},  
"playback": {  
  "in":1.5,  
  "out":10.0  
}  
}
```

If you download an existing video content item using the web service, or export one, you may see other fields in the data structure that are added by the Content Store after a content item has been created, but you do not need to supply values for these fields when creating content items.

6 Accessing Media from Templates

You can access the transcoded versions of a video/audio clip from your JSP templates using JSTL as follows (in all the examples *video-field-name* is the name of the video field in your content type and *audio-field-name* is the name of the audio field in your content type and *index* is the index of the transcoded version you want).

Note that if a video's transcoded outputs include an HLS stream, then the number of transcoded versions will not match the number of presets in the selected preset group, since all HLS presets are merged and result in only one HLS stream. For further information about this, see [section 3.5.1](#).

- To access the URI of a transcoded video:

```
| ${article.fields.video-field-name.value.video[index].uri}
```

- To access the video's MIME type:

```
| ${article.fields.video-field-name.value.video[index]['mime-type']}
```

HLS streams generated by Elastic Transcoder are given the MIME type **application/x-mpegURL**.

- To access the video's height:

```
| ${article.fields.video-field-name.value.video[index].height}
```

- To access the video's width:

```
| ${article.fields.video-field-name.value.video[index].width}
```

- To access the video's duration:

```
| ${article.fields.video-field-name.value.duration}
```

- To access the video's timeline information:

```
| ${article.fields.video-field-name.value.timeline.cuepoints}
```

```
| ${article.fields.video-field-name.value.timeline.tracks}
```

```
| ${article.fields.video-field-name.value.playback}
```

- To access the video's key frames:

```
| ${article.relatedElements.KEYFRAMES.items}
```

If a poster frame is selected than the first key frame will be the poster frame.